GE Fanuc Automation

Programmable Control Products

Series 90-30/20/Micro Programmable Controllers

Reference Manual

GFK-0467K *September 1998*

Warnings, Cautions, and Notes as Used in this Publication

Warning

Warning notices are used in this publication to emphasize that hazardous voltages, currents, temperatures, or other conditions that could cause personal injury exist in this equipment or may be associated with its use.

In situations where inattention could cause either personal injury or damage to equipment, a Warning notice is used.

Caution

Caution notices are used where equipment might be damaged if care is not taken.

Note

Notes merely call attention to information that is especially significant to understanding and operating the equipment.

This document is based on information available at the time of its publication. While efforts have been made to be accurate, the information contained herein does not purport to cover all details or variations in hardware or software, nor to provide for every possible contingency in connection with installation, operation, or maintenance. Features may be described herein which are not present in all hardware and software systems. GE Fanuc Automation assumes no obligation of notice to holders of this document with respect to changes subsequently made.

GE Fanuc Automation makes no representation or warranty, expressed, implied, or statutory with respect to, and assumes no responsibility for the accuracy, completeness, sufficiency, or usefulness of the information contained herein. No warranties of merchantability or fitness for purpose shall apply.

The following are trademarks of GE Fanuc Automation North America, Inc.

Alarm Master	GEnet	PowerMotion	Series One
CIMPLICITY	Genius	ProLoop	Series Six
CIMPLICITY PowerTRAC	Genius PowerTRAC	PROMACRO	Series Three
CIMPLICITY 90–ADS	Helpmate	Series Five	VuMaster
CIMSTAR	Logicmaster	Series 90	Workmaster
Field Control	Modelmaster	VersaMax	

All Rights Reserved.

This manual describes the system operation, fault handling, and Logicmaster 90™ programming instructions for the Series 90™-30, Series 90™-20 and Series 90™ Micro programmable logic controllers. Series 90-30 PLCs, Series 90-20 PLCs, and Series 90 Micro PLCs are all members of the Series 90™ family of programmable logic controllers from GE Fanuc Automation.

Revisions to This Manual

There are new 350 and 360 series CPUs. Differences in their memory limits and general operations are specified in Chapter 2 of this manual, "System Operations."
There are two new Service Requests and one new function in Release 9.02 of Logicmaster. Service Request # 46, Fast Backplane Status Access, is discussed in Chapter 4, on page 4-165 and following. The Sequential Event Recorder is a new function and discussed on page 4-114 and following. There are also new fault reported from some of the newer CPUs. Descriptions and corrections for those faults are discussed on page 3-8 and following.
Beginning with Release 8, the 352 CPU supported floating-point operations. Beginning with Release 9, the 350, 351, 360, 363, and 364 CPUs also support floating-point operations, but there are some differences between the software floating-point capabilities of those models and the floating-point capabilities of the 352 CPU which uses a floating-point math co-processor. Those differences are discussed in Appendix E. Also, the instructional timing information in Appendix A includes floating-point and other instructional timing for these new models.

Content of This Manual

Chapter 1. Introduction: provides an overview of the Series 90-30 PLC, the Series 90-20 PLC, and the Series 90 Micro PLC systems and the Series 90-30/20/Micro instruction set.

Chapter 2. System Operation: describes certain system operations of the Series 90-30 PLC, Series 90-20 PLC, or Series 90 Micro systems. This includes a discussion of the PLC system sweep sequences, the system power-up and power-down sequences, clocks and timers, security, I/O, and fault handling. It also includes general information for a basic understanding of programming ladder logic.

Chapter 3. Fault Explanations and Correction: provides troubleshooting information for a Series 90-30, 90-20, or Micro PLC system. It explains fault descriptions in the PLC fault table and fault categories in the I/O fault table.

Chapter 4. Series 90-30/20/Micro Instruction Set: describes programming instructions available for Series 90-30 PLCs, Series 90-20 PLCs, Series 90 Micro PLCs. The information in this chapter is arranged as sections that correspond to the main program function groups.

Appendix A. Instruction Timing: lists the memory size in bytes and execution time in microseconds for each programming instruction. Memory size is the number of bytes required by the function in a ladder diagram application program.

Appendix B. Interpreting Fault Tables: describes how to interpret the message structure format when reading the fault tables using Logicmaster 90-30/20/Micro software.

Appendix C. Instruction Mnemonics: lists mnemonics that can be typed to display programming instructions while searching through or editing a program.

Appendix D. Key Functions: lists the special keyboard assignments used for the Logicmaster 90-30/20/Micro software.

Appendix E. Using Floating-Point Numbers: describes special considerations for using floating-point math operations.

Related Publications

Logicmaster™ 90 Series 90™-30/20/Micro Programming Software User's Manual

Logicmaster™ 90 Series 90-30 and 90-20 Important Product Information

Series 90™-30 Programmable Controller Installation Manual

Series 90™-20 Programmable Controller Installation Manual (GFK-0551).

Series 90™-30 I/O Module Specifications Manual (GFK-0898).

Series 90™ Programmable Coprocessor Module and Support Software User's Manual (GFK-0255).

Series 90™ PCM Development Software (PCOP) User's Manual

CIMPLICITY™ 90-ADS Alphanumeric Display System User's Manual

CIMPLICITY™ 90-ADS Alphanumeric Display System Reference Manual

Alphanumeric Display Coprocessor Module Data Sheet (GFK-0521).

Series 90™-30 and 90-20 PLC Hand-Held Programmer User's Manual

Power Mate APM for Series 90™-30 PLC—Standard Mode User's Manual (GFK-0840).

Power Mate APM for Series 90™-30 PLC—Follower Mode User's Manual (GFK-0781).

Series 90™-30 High Speed Counter User's Manual

Series 90™-30 Genius Communications Module User's Manual

Genius Communications Module Data Sheet (GFK-0272).

Series 90™-30 Genius™ Bus Controller User's Manual (GFK-1034).

Series 90™-70 FIP Bus Controller User's Manual

Series 90™-30 FIP Remote I/O Scanner User's Manual

Field Control™ Distributed I/O and Control System Genius™ Bus Interface Unit User's Manual

Series 90™ Micro Programmable Logic Controller User's Manual

Series 90™ PLC Serial Communications User's Manua

We Welcome Your Comments and Suggestions

At GE Fanuc Automation, we strive to produce quality technical documentation. After you have used this manual, please take a few moments to complete and return the Reader's Comment Card located on the next page.

David D. Bruton

Sr. Technical Writer

Contents

Chapter 1 Introduction 1-1

Additional Reference Information: See the appendices in the back of this manual....1-2

Chapter 2 System Operation 2-1

Section 1: PLC Sweep Summary 2-2

Standard Program Sweep 2-2

Sweep Time Calculation..... 2-6

Housekeeping..... 2-6

Input Scan 2-7

Application Program Logic Scan or Solution..... 2-8

Output Scan..... 2-9

Logic Program Checksum Calculation..... 2-9

Programmer Communications Window..... 2-9

System Communications Window (Models 331 and Higher) 2-10

PCM Communications with the PLC (Models 331 and Higher)..... 2-12

Standard Program Sweep Variations 2-13

Constant Sweep Time Mode 2-13

PLC Sweep When in STOP Mode 2-13

Communication Window Modes..... 2-14

Key Switch on 350 and 360 Series CPUs: Change Mode and Flash Protect 2-14

Using the Release 7 and Later Key Switch 2-14

Clearing the Fault Table with the Key Switch 2-14

Enhanced Memory Protect with Release 8 and Later CPUs 2-15

Section 2: Program Organization and User References/Data..... 2-16

Subroutine Blocks (Series 90-30 PLC only)..... 2-16

Examples of Using Subroutine Blocks..... 2-18

How Blocks Are Called..... 2-19

Periodic Subroutines..... 2-19

User References 2-20

Transitions and Overrides..... 2-21

Retentiveness of Data 2-21

Data Types 2-23

System Status References 2-24

Function Block Structure 2-26

Format of Ladder Logic Relays..... 2-26

Format of Program Function Blocks 2-27

Function Block Parameters..... 2-28

Power Flow In and Out of a Function 2-29

Section 3: Power-Up and Power-Down Sequences 2-30

Power-Up 2-30

Power-Down 2-33

Section 4: Clocks and Timers..... 2-34

Elapsed Time Clock 2-34
Time-of-Day Clock 2-34
Watchdog Timer 2-35
Constant Sweep Timer 2-35
Time-Tick Contacts 2-35

Section 5: System Security 2-36
Passwords 2-36
Privilege Level Change Requests 2-37
Locking/Unlocking Subroutines 2-37
Permanently Locking a Subroutine 2-37

Section 6: Series 90-30, 90-20, and Micro I/O System 2-38
Model 30 I/O Modules 2-39
I/O Data Formats 2-41
Default Conditions for Model 30 Output Modules 2-41
Diagnostic Data 2-41
Global Data 2-42
Model 20 I/O Modules 2-42
Micro PLCs 2-43

Chapter 3 Fault Explanation and Correction 3-1

Section 1: Fault Handling 3-2
Alarm Processor 3-2
Classes of Faults 3-2
System Reaction to Faults 3-3
Fault Tables 3-3
Fault Action 3-4
Fault References 3-4
Fault Reference Definitions 3-5
Additional Fault Effects 3-5
PLC Fault Table Display 3-5
I/O Fault Table Display 3-5
Accessing Additional Fault Information 3-6

Section 2: PLC Fault Table Explanations 3-7
Fault Actions 3-8
Loss of, or Missing, Option Module 3-8
Reset of, Addition of, or Extra, Option Module 3-9
System Configuration Mismatch 3-10
Option Module Software Failure 3-11
Program Block Checksum Failure 3-11
Low Battery Signal 3-11
Constant Sweep Time Exceeded 3-12

Application Fault 3-12
No User Program Present 3-13
Corrupted User Program on Power-Up 3-13
Password Access Failure 3-13
PLC CPU System Software Failure 3-14
Communications Failure During Store 3-16

Section 3: I/O Fault Table Explanations 3-17
Loss of I/O Module 3-17
Addition of I/O Module 3-18

Chapter 4 Series 90-30/20/Micro Instructions Set 4-1

Section 1: Relay Functions 4-2
Using Contacts 4-2
Using Coils 4-3
Normally Open Contact —| |— 4-4
Normally Closed Contact —|/|— 4-4
Example: 4-4
Coil —( )— 4-4
Example: 4-4
Negated Coil —(/)— 4-4
Example: 4-4
Retentive Coil —(M)— 4-5
Negated Retentive Coil —(/M)— 4-5
Positive Transition Coil —(↑)— 4-5
Negative Transition Coil —(↓)— 4-5
Example: 4-5
SET Coil —(S)— 4-6
RESET Coil —(R)— 4-6
Example: 4-6
Retentive SET Coil —(SM)— 4-7
Retentive RESET Coil —(RM)— 4-7
Links 4-7
Example: 4-7
Continuation Coils (———<+>) and Contacts (<+>———) 4-8

Section 2: Timers and Counters 4-9
Function Block Data Required for Timers and Counters 4-9
ONDTR 4-11
Parameters: 4-12
Valid Memory Types: 4-12
Example: 4-13
TMR 4-14
Parameters: 4-15
Valid Memory Types: 4-15

Example: 4-16
OFDT....4-17
Parameters:....4-18
Valid Memory Types: 4-19
Example: 4-19
UPCTR 4-20
Parameters:....4-20
Valid Memory Types: 4-21
Example: 4-21
DNCTR....4-22
Parameters:....4-22
Valid Memory Types: 4-23
Example: 4-23
Example: 4-24
Section 3: Math Functions....4-26
Standard Math Functions (ADD, SUB, MUL, DIV)....4-27
Parameters:....4-28
Valid Memory Types: 4-28
Example: 4-28
Math Functions and Data Types....4-29
MOD (INT, DINT) 4-31
Parameters:....4-31
Valid Memory Types: 4-32
Example: 4-32
SQRT (INT, DINT, REAL)....4-33
Parameters:....4-33
Valid Memory Types: 4-34
Example: 4-34
Trig Functions (SIN, COS, TAN, ASIN, ACOS, ATAN) 4-35
Parameters:....4-36
Valid Memory Types: 4-36
Example: 4-36
Logarithmic/Exponential Functions (LOG, LN, EXP, EXPT) 4-37
Parameters:....4-37
Valid Memory Types: 4-38
Example: 4-38
Radian Conversion (RAD, DEG) 4-39
Parameters:....4-39
Valid Memory Types: 4-40
Example: 4-40
Section 4: Relational Functions....4-41
Parameters:....4-42
Expanded Description:....4-42
Valid Memory Types: 4-42
Example: 4-43
RANGE (INT, DINT, WORD) 4-44
Parameters:....4-45
Valid Memory Types: 4-45
Example 1: 4-46

Example 2: 4-46
Section 5: Bit Operation Functions 4-47
AND and OR (WORD) 4-49
Parameters: 4-49
Valid Memory Types: 4-50
Example: 4-50
XOR (WORD) 4-51
Parameters: 4-51
Valid Memory Types: 4-52
Example: 4-52
NOT (WORD) 4-53
Parameters: 4-53
Valid Memory Types: 4-54
Example: 4-54
SHL and SHR (WORD) 4-55
Parameters: 4-56
Valid Memory Types: 4-56
Example: 4-57
ROL and ROR (WORD) 4-58
Parameters: 4-58
Valid Memory Types: 4-59
Example: 4-59
BTST (WORD) 4-60
Parameters: 4-60
Valid Memory Types: 4-61
Example: 4-61
BSET and BCLR (WORD) 4-62
Parameters: 4-62
Valid Memory Types: 4-63
Example: 4-63
BPOS (WORD) 4-64
Parameters: 4-64
Valid Memory Types: 4-65
Example: 4-65
MSKCMP (WORD, DWORD) 4-66
If All Bits in I1 and I2 are the Same 4-66
If a Miscompare is Found 4-66
Parameters: 4-67
Valid Memory Types: 4-67
Example: 4-68
Section 6: Data Move Functions 4-69
MOVE (BIT, INT, WORD, REAL) 4-70
Parameters: 4-71
Valid Memory Types: 4-71
Example 1: 4-72
Example 2: 4-72
BLKMOV (INT, WORD, REAL) 4-73
Parameters: 4-73

Valid Memory Types: 4-74
Example: 4-74
BLKCLR (WORD)....4-75
Parameters:....4-75
Valid Memory Types: 4-76
Example: 4-76
SHFR (BIT, WORD)....4-77
Parameters:....4-78
Valid Memory Types: 4-78
Example 1: 4-79
Example 2: 4-79
BITSEQ (BIT)....4-80
Memory Required for a Bit Sequencer 4-80
Parameters:....4-81
Valid Memory Types: 4-82
Example: 4-82
COMMREQ....4-83
Command Block 4-83
Parameters:....4-84
Valid Memory Types: 4-84
Example: 4-85
Section 7: Table Functions 4-86
ARRAY_MOVE (INT, DINT, BIT, BYTE, WORD)....4-87
Parameters:....4-88
Valid Memory Types: 4-88
Example 1: 4-89
Example 2: 4-89
Example 3: 4-90
SRCH_EQ and SRCH_NE (INT, DINT, BYTE, WORD) SRCH_GT and SRCH_LT SRCH_GE and SRCH_LE....4-91
Parameters:....4-92
Valid Memory Types: 4-92
Example 1: 4-93
Example 2: 4-93
Section 8: Conversion Functions....4-94
—>BCD-4 (INT)....4-95
Parameters:....4-95
Valid Memory Types: 4-96
Example: 4-96
—>INT (BCD-4, REAL)....4-97
Parameters:....4-97
Valid Memory Types: 4-98
Example: 4-98
—>DINT (REAL)....4-99
Parameters:....4-99
Valid Memory Types: 4-100
Example: 4-100
—>REAL (INT, DINT, BCD-4, WORD)....4-101

Parameters:....4-101
Valid Memory Types:....4-102
Example:....4-102
—>WORD (REAL)....4-103
Parameters:....4-103
Valid Memory Types:....4-104
Example:....4-104
TRUN (INT, DINT)....4-105
Parameters:....4-105
Valid Memory Types:....4-106
Example:....4-106
Section 9: Control Functions....4-107
CALL....4-108
Example:....4-108
DOIO....4-109
Parameters:....4-110
Valid Memory Types:....4-110
Input Example 1:....4-111
Input Example 2:....4-111
Output Example 1:....4-112
Output Example 2:....4-112
Enhanced DO I/O Function for 331 and Later CPUs....4-113
SER....4-114
Parameters:....4-114
Valid Memory Types:....4-114
Status Extra Data....4-117
SER Data Block....4-118
SER Notes....4-118
Example:....4-120
Data Block....4-122
END....4-123
Example:....4-123
MCR....4-124
Differences Between MCRs and JUMPs....4-125
Example:....4-126
ENDMCR....4-127
Example:....4-127
JUMP....4-128
Example:....4-129
LABEL....4-130
Example:....4-130
COMMENT....4-131
SVCREQ....4-132
Parameters:....4-133
Valid Memory Types:....4-133
Example:....4-133
SVCREQ #1: Change/Read Constant Sweep Timer....4-134
Example:....4-136

SVCREQ #2: Read Window Values 4-137
SVCREQ #3: Change Prog.Communications Window Mode & Timer Value 4-139
Example: 4-140
SVCREQ #4: Change System Comm. Window Mode and Timer Value 4-141
SVCREQ #6: Change/Read Number of Words to Checksum 4-143
To Read the Current Word Count: 4-143
To Set a New Word Count: 4-143
Example: 4-144
SVCREQ #7: Change/Read Time-of-Day Clock 4-145
Example: 4-146
Parameter Block Contents 4-147
To Change/Read Date and Time Using BCD Format: 4-147
To Change/Read Date and Time using Packed ASCII with Embedded Colons Format 4-148
SVCREQ #8: Reset Watchdog Timer 4-149
Example: 4-149
SVCREQ #9: Read Sweep Time from Beginning of Sweep 4-150
Example: 4-150
SVCREQ #10: Read Folder Name 4-151
Example: 4-151
SVCREQ #11: Read PLC ID 4-152
Example: 4-152
SVCREQ #12: Read PLC Run State 4-153
Example: 4-153
SVCREQ #13: Shut Down (Stop) PLC 4-154
Example: 4-154
SVCREQ #14: Clear Fault Tables 4-155
Example: 4-155
SVCREQ #15: Read Last-Logged Fault Table Entry 4-156
Example 1: 4-157
Example 2: 4-158
SVCREQ #16: Read Elapsed Time Clock 4-160
Example: 4-160
SVCREQ #18: Read I/O Override Status 4-161
Example: 4-161
SVCREQ #23: Read Master Checksum 4-162
Example: 4-162
SVCREQ #26/30: Interrogate I/O 4-163
Example: 4-163
SVCREQ #29: Read Elapsed Power Down Time 4-164
Example: 4-164
SVCREQ #46:Fast Backplane Status Access 4-165
Read Extra Status Data (Function #1) 4-165
Write Data (Function #2) 4-167
Read/Write Data (Function #3) 4-168
Example 1: 4-169
Example 2: 4-170
PID 4-171
Parameters: 4-172
Valid Memory Types: 4-172

PID Parameter Block: 4-173
Operation of the PID Instruction....4-175
Internal Parameters in RefArray....4-178
PID Algorithm Selection (PIDISA or PIDIND) and Gains....4-179
CV Amplitude and Rate Limits....4-180
Sample Period and PID Block Scheduling....4-181
Determining the Process Characteristics....4-181
Setting User Parameters Including Tuning Loop Gains....4-182
Setting Loop Gains — Ziegler and Nichols Tuning Approach....4-183
Sample PID Call....4-184

Appendix A Instruction Timing....A-1

Boolean Execution Speed....A-10

Appendix B Interpreting Fault Tables....B-1

PLC Fault Table....B-2
I/O Fault Table....B-8

Appendix C Instruction Mnemonics....C-1

Appendix D Key Functions....D-1

Appendix E Using Floating-Point Numbers....E-1

Floating-Point Numbers....E-1
Internal Format of Floating-Point Numbers....E-3
Values of Floating-Point Numbers....E-4
Entering and Displaying Floating-Point Numbers....E-5
Errors in Floating-Point Numbers and Operations....E-6

Figure 2-1. PLC Sweep....2-3
Figure 2-2. Programmer Communications Window Flow Chart....2-10
Figure 2-3. System Communications Window Flow Chart....2-11
Figure 2-4. PCM Communications with the PLC....2-12
Figure 2-5. Power-Up Sequence....2-31
Figure 2-6. Time-Tick Contact Timing Diagram....2-35
Figure 2-7. Series 90-30 I/O Structure....2-38
Figure 2-8. Model 30 I/O Modules....2-39
Independent Term Algorithm (PIDIND)....4-180

Table 2-1. Sweep Time Contribution 2-4
Table 2-2. I/O Scan Time Contributions for the 90-30 350 and 360 Series (in milliseconds) 2-5
Table 2-3. Register References 2-20
Table 2-4. Discrete References 2-20
Table 2-4. Discrete References - Continued 2-21
Table 2-5. Data Types 2-23
Table 2-6. System Status References 2-24
Table 2-6. System Status References - Continued 2-25
Table 2-6. System Status References - Continued 2-26
Table 2-7. Model 30 I/O Modules - Continued 2-40
Table 2-7. Model 30 I/O Modules - Continued 2-41
Table 3-1. Fault Summary 3-3
Table 3-2. Fault Actions 3-4
Table 4-1. Types of Contacts 4-2
Table 4-2. Types of Coils 4-3
Table 4-3. Service Request Functions 4-132
Table 4-4. PID Parameters Overview 4-173
Table 4-4. PID Parameters Overview (Continued) 4-174
Table 4-5. PID Parameters Details 4-176
Table 4-5. PID Parameters Details - Continued 4-177
Table 4-5. PID Parameters Details - Continued 4-178
Table A-1. Instruction Timing A-2
Table A-1. Instruction Timing-Continued A-3
Table A-1. Instruction Timing-Continued A-4
Table A-1. Instruction Timing-Continued A-5
Table A-1. Instruction Timing-Continued A-6
Table A-1. Instruction Timing-Continued A-7
Table A-1. InstructionTiming-Continued A-8
Table A-1. Instruction Timing-Continued A-9
Table A-2. Instruction Sizes for 350 and 360 Series CPUs A-10
Table A-3. Boolean Execution Speeds A-10
Table B-1. PLC Fault Groups B-4
Table B-2. PLC Fault Actions B-5
Table B-3. Alarm Error Codes for PLC CPU Software Faults B-5
Table B-4. Alarm Error Codes for PLC Faults B-6

Table B-5. PLC Fault Data - Illegal Boolean Opcode Detected B-7
Table B-6. PLC Fault Time Stamp B-7
Table B-7. I/O Fault Table Format Indicator Byte B-9
Table B-8. I/O Reference Address B-9
Table B-9. I/O Reference Address Memory Type B-9
Table B-10. I/O Fault Groups B-10
Table B-11. I/O Fault Actions B-11
Table B-12. I/O Fault Specific Data B-11
Table B-13. I/O Fault Time Stamp B-12
General Case of Power Flow for Floating-Point Operations E-7

Chapter 1 Introduction

The Series 90-30, 90-20, and Micro PLCs are members of the GE Fanuc Series 90™ family of Programmable Logic Controllers (PLCs). They are easy to install and configure, offer advanced programming features, and are compatible with the Series 90-70 PLC.

The Series 90-20 PLC provides a cost-effective platform for low I/O count applications. The primary objectives of the Series 90-20 PLC are as follows:

- To provide a small PLC that is easy to use, install, upgrade, and maintain.
- To provide a cost-effective family-compatible PLC.
- To provide easier system integration through standard communication hardware and protocols.

The Series 90 Micro PLC also provides a cost-effective platform for lower I/O count applications. The primary objectives of the Micro PLC are the same as those for the Series 90-20. In addition, the Micro offers the following:

- The Micro PLC has the CPU, power supply, inputs and outputs all built into one small device.
- Most models also have a high speed counter.
- Because the CPU, power supply, inputs and outputs all built into one device, it is very easy to configure.

The software structure for the Series 90-30 PLC (except the 350 and higher models) and Series 90-20 PLC uses an architecture that manages memory and execution priority in the 80188 microprocessor. The 350 and 360 series of 90-30 PLCs use an 80386EX microprocessor. The Series 90 Micro PLC uses the H8 microprocessor. This operation supports both program execution and basic housekeeping tasks such as diagnostic routines, input/output scanners, and alarm processing. The system software also contains routines to communicate with the programmer. These routines provide for the upload and download of application programs, return of status information, and control of the PLC.

In the Series 90-30 PLC, the application (user logic) program that controls the end process to which the PLC is applied is controlled by a dedicated Instruction Sequencer Coprocessor (ISCP). The ISCP is implemented in hardware in the Model 313 and higher and in software in the Model 311 systems, and the Micro PLC. The 80188 microprocessor and the ISCP can execute simultaneously, allowing the microprocessor to service communications while the ISCP is executing the bulk of the application program; however, the microprocessor must execute the non-boolean function blocks.

Faults occur in the Series 90-30 PLC, Series 90-20 PLC, and the Micro PLC when certain failures or conditions happen that affect the operation and performance of the system. These conditions may affect the ability of the PLC to control a machine or process. Other conditions may only act as

an alert, such as a low battery signal to indicate that the voltage of the battery protecting the memory is low and should be replaced. The condition or failure is called a fault.

Faults are handled by a software alarm processor function that records the faults in either the PLC fault table or the I/O fault table. (The Model 331 and Model 340/341 CPUs also time-stamp the faults.) These tables can be displayed through the programming software on the PLC Fault Table and I/O Fault Table screens in Logicmaster 90-30/20/Micro software using the control and status functions.

Additional Reference Information: See the appendices in the back of this manual.

Appendix A lists the memory size in bytes and the execution time in microseconds for each programming instruction.

Appendix B describes how to interpret the message structure format when reading the PLC and I/O fault tables.

Appendix C lists the instruction mnemonics used with Logicmaster 90-30/20/Micro software.

Appendix D lists the special keyboard assignments used with Logicmaster 90-30/20/Micro software.

Appendix E provides special considerations and instructions for using floating-point math (available only on the 350 and 360 series of 90-30 CPUs).

Chapter 2 System Operation

This chapter describes certain system operations of the Series 90-30, 90-20, and Micro PLC systems. These system operations include:

- A summary of PLC sweep sequences (see Section 1).
- Program organization and user references/data (see Section 2).
- Power-up and power-down sequences (see Section 3).
- Clocks and timers (see Section 4).
- System security through password assignment (see Section 5).
- Model 30 I/O modules (see Section 6).

Section 1: PLC Sweep Summary

The logic program in the Series 90-30, 90-20, and Micro PLCs execute repeatedly until stopped by a command from the programmer or a command from another device. The sequence of operations necessary to execute a program one time is called a sweep. In addition to executing the logic program, the sweep includes obtaining data from input devices, sending data to output devices, performing internal housekeeping, servicing the programmer, and servicing other communications.

Series 90-30, 90-20, and Micro PLCs normally operate in **STANDARD PROGRAM SWEEP** mode. Other operating modes include **STOP WITH I/O DISABLED** mode, **STOP WITH I/O ENABLED** mode, and **CONSTANT SWEEP** mode. Each of these modes described in this chapter is controlled by external events and application configuration settings. The PLC makes the decision regarding its operating mode at the start of every sweep.

Standard Program Sweep

STANDARD PROGRAM SWEEP mode normally runs under all conditions. The CPU operates by executing an application program, updating I/O, and performing communications and other tasks. This occurs in a repetitive cycle called the CPU sweep. There are seven parts to the execution sequence of the Standard Program Sweep:

1. Start-of-sweep housekeeping
2. Input scan (read inputs)
3. Application program logic solution
4. Output scan (update outputs)
5. Programmer service
6. Non-programmer service
7. Diagnostics

All of these steps execute every sweep. Although the Programmer Communications Window opens each sweep, programmer services only occur if a board fault has been detected or if the programming device issues a service request; that is, the Programmer Communications Window first checks for work to do and exits if there is none. The sequence of the standard program sweep is shown in the following figure.

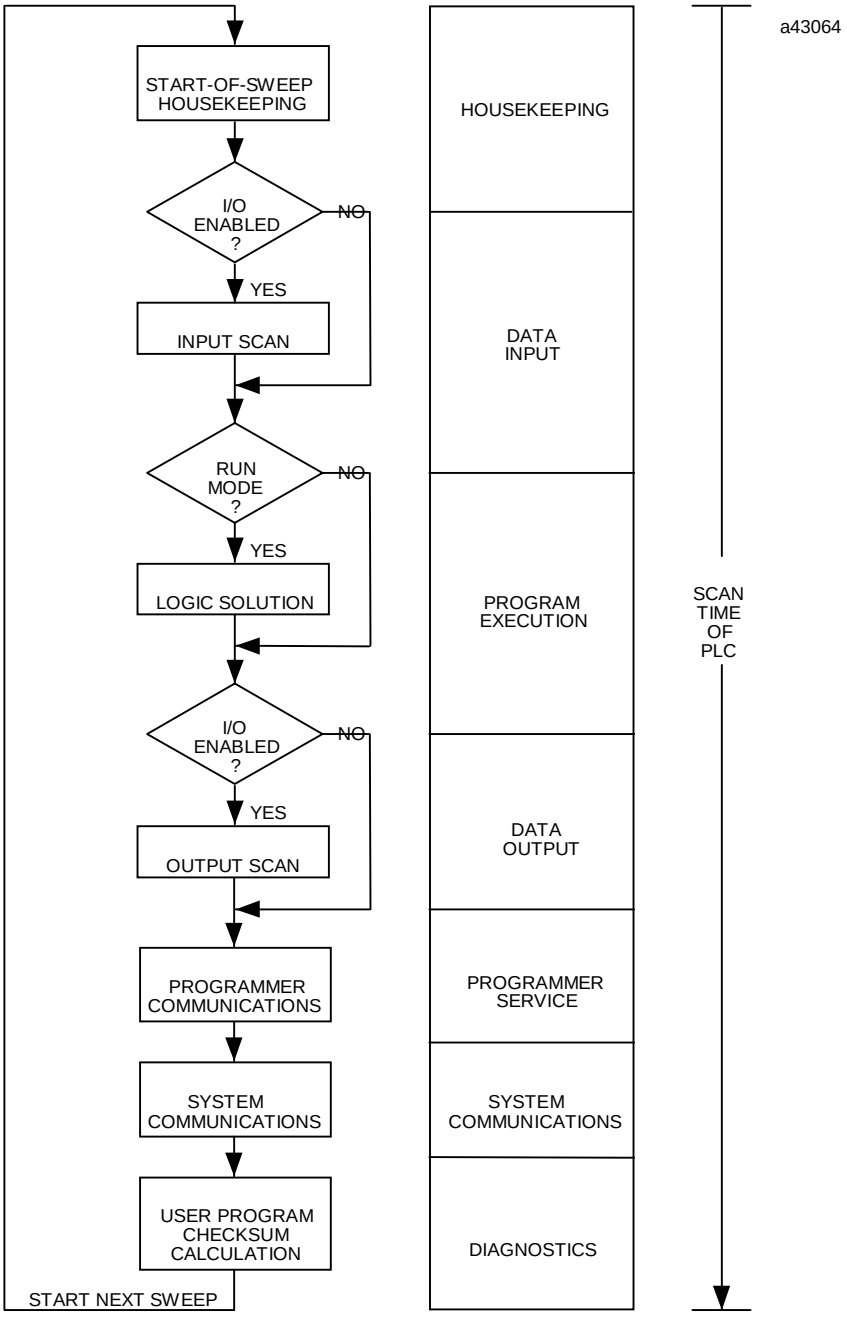


Figure 2-1. PLC Sweep

As shown in the PLC sweep sequence, several items are included in the sweep. These items contribute to the total sweep time as shown in the following table.

Table 2-1. Sweep Time Contribution

Sweep Element	Description		Time Contribution (ms) [4]
			351/352 (350 and 360 series—see note)
Housekeeping	• Calculate sweep time. • Schedule start of next sweep. • Determine mode of next sweep. • Update fault reference tables. • Reset watchdog timer.		0.279
Data Input	Input data is received from input and option modules.		See Table 2-2 for scan time contributions.
Program Execution	User logic is solved.		Execution time is dependent upon the length of the program and the type of instructions used in the program. Instruction execution times are listed in Appendix A.
Data Output	Output data is sent to output and option modules.		See Table 2-2 for scan time contributions.
Service External Devices	Service requests from programming devices and intelligent modules are processed. [1]	HHP	0.334
		LM-90	0.517
		PCM [2]	0.482
Reconfiguration	Slots with faulted modules and empty slots are monitored.		0.319
Diagnostics	Verify user program integrity		0.010 per word checksummed each sweep[3]

1. The scan time contribution of external device service is dependent upon the mode of the communications window in which the service is processed. If the window mode is **`LIMITED`**, a maximum of 8 milliseconds for the 311, 313, 323, and 331 CPUs and 6 milliseconds for the 340 and higher CPUs will be spent during that window. If the window mode is **`RUN-TO-COMPLETION`**, a maximum of 50 ms can be spent in that window, depending upon the number of requests which are presented simultaneously.
2. These measurements were taken with the PCM physically present but not configured and with no application task running on the PCM.
3. The number of words checksummed each sweep can be changed with the SVCREQ function block.
4. These measurements were taken with an empty program and the default configuration. The Series 90-30 PLCs were in an empty 10-slot rack with no extension racks connected. Also, the times in this table assume that there is no periodic subroutine active; the times will be larger if a periodic subroutine is active.
5. The data input time for the Micro PLC can be determined as follows: 0.365 ms. (fixed scan) + 0.036 ms. (filter time) x (total sweep time)/0.5 ms.
6. Since the Micro PLC has a static set of I/O, reconfiguration is not necessary.
7. Since the user program for the Micro PLC is in Flash memory, it will not be checked for integrity.

Note

The times for the 350 CPU and the 360 series are estimated to be the same.

Table 2-2. I/O Scan Time Contributions for the 90-30 350 and 360 Series (in milliseconds)

Module Type		CPU		
		350 and 360 Series		
		Main Rack	Expansion Rack	Remote Rack
8-point discrete input		.030	.055	.206
16-point discrete input		.030	.055	.206
32-point discrete input		.043	.073	.269
8-point discrete output		.030	.053	.197
16-point discrete output		.030	.053	.197
32-point discrete output		.042	.070	.259
Combination discrete input/output		.060	.112	.405
4-channel analog input		.075	.105	.396
2-channel analog output		.058	.114	.402
16-channel analog input (current or voltage)		.978	1.446	3.999
8-channel analog output		1.274	1.988	4.472
Combination analog input/output		1.220	1.999	4.338
High Speed Counter		1.381	2.106	5.221
Power Mate APM (1-axis)		1.527	2.581	6.388
I/O Processor		1.574	2.402	6.388
Ethernet Interface (no connection)		.038	.041	.053
GCM	no devices	.911	1.637	5.020
	8 64-word devices	8.826	16.932	21.179
GCM+	no devices	.567	.866	1.830
	32 64-word devices	1.714	2.514	5.783
GBC	no devices	.798	1.202	2.540
	32 64-word devices	18.382	25.377	70.777
PCM 311	not configured, or no application task	.476	N/A	N/A
	read 128 %R as fast as possible	.485	N/A	N/A
ADC (no task)		.476	N/A	N/A
I/O Link Master	no devices	.569	.865	1.932
	16 64-point devices	4.948	7.003	19.908
I/O Link Slave	32-point	.087	.146	.553
	64-point	.154	.213	.789

Sweep Time Calculation

Table 2-1 lists the seven items that contribute to the sweep time of the PLC. The sweep time consists of fixed times (housekeeping and diagnostics) and variable times. Variable times vary according to the I/O configuration, size of the user program, and the type of programming device connected to the PLC.

Example of Sweep Time Calculation

An example of the calculations for determining the sweep time for a Series 90-30 model 331 PLC are shown in the table shown below.

The modules and instructions used for these calculations are listed below:

- Input modules: five 16-point model 30 input modules.
- Output modules: four 16-point model 30 output modules.
- Programming instructions: A 1200-step program consisting of 700 boolean instructions (LD, AND, OR, etc.), 300 output coils (OUT, OUTM, etc.), and 200 math functions (ADD, SUB, etc.).

Sweep Component	Calculation	Time Contribution		
		wo/ Programmer	w/ HHP	w/ LM90
Housekeeping	0.705 ms	0.705 ms	0.705 ms	0.705 ms
Data Input	0.055 x 5 = .275 ms	0.275 ms	0.275 ms	0.275 ms
Program Execution	1000 x 0.4 µs* + 200 x 89 µs** = 18.2 ms	18.2 ms	18.2 ms	18.2 ms
Data Output	0.061 x 4 = .244 ms	0.244 ms	0.244 ms	0.244 ms
Programmer Service	0.4 ms + programmer time + 0.6 ms	0 ms	4.524 ms	2.454 ms
Non-Programmer Service	None in this example	0 ms	0 ms	0 ms
Reconfiguration	0.639 ms	0.639 ms	0.639 ms	0.638 ms
Diagnostics	0.048 ms	0.048 ms	0.048 ms	0.048 ms
PLC Sweep Time	Housekeeping + Data Input + Program Execution + Data Output + Programmer Service + Non-Programmer Service + Diagnostics	12.611 ms	17.135 ms	15.065 ms

* Refer to Table A-3, "Boolean Execution Speed," in Appendix A for contact and coil execution speeds, which vary by CPU.

** Refer to Table A-1, "Instruction Timing," for specific timing information. The figure used in this calculation, 89 microseconds, represents the average of the add, subtract, shift, used in the example.

Housekeeping

The housekeeping portion of the sweep performs all of the tasks necessary to prepare for the start of the sweep. If the PLC is in **`CONSTANT SWEEP`** mode, the sweep is delayed until the required sweep time elapses. If the required time has already elapsed, the OV_SWP %SA0002 contact is set, and the sweep continues without delay. Next, timer values (hundredths, tenths, and seconds) are updated by calculating the difference from the start of the previous sweep and the new sweep

time. In order to maintain accuracy, the actual start of sweep is recorded in 100 microsecond increments. Each timer has a remainder field which contains the number of 100 microsecond increments that have occurred since the last time the timer value was incremented.

Input Scan

Scanning of inputs occurs during the input scan portion of the sweep, just prior to the logic solution. During this part of the sweep, all Model 30 input modules are scanned and their data stored in %I (discrete inputs) or %AI (analog inputs) memory, as appropriate. Any global data input received by a Genius Communications Module, an Enhanced Genius Communications Module, or a Genius Bus Controller is stored in %G memory.

Modules are scanned in ascending reference address order, starting with the Genius Communications Module, then discrete input modules, and finally analog input modules.

If the CPU is in **STOP** mode and the CPU is configured to not scan I/O in **STOP** mode, the input scan is skipped.

Application Program Logic Scan or Solution

The application program logic scan is when the application logic program actually executes. The logic solution always begins with the first instruction in the user application program immediately following the completion of the input scan. Solving the logic provides a new set of outputs. The logic solution ends when the END instruction is executed (the END is invisible unless you are using a Hand-Held Monitor).

The application program is executed by the ISCP and the 80C188 microprocessor. In the Model 313 and higher CPUs, the ISCP executes the boolean instructions; and the 80C188 or 80386EX executes the timer, counter, and function blocks. In the Model 311 and 90-20 CPUs, the 80C188 executes all boolean, timer, counter, and function block instructions. On the Micro, the H8 processor executes all booleans and function blocks.

A list of execution times for each programming function can be found in Appendix A.

Output Scan

Outputs are scanned during the output scan portion of the sweep, immediately following the logic solution. Outputs are updated using data from %Q (for discrete outputs) and %AQ (for analog outputs) memory, as appropriate. If the Genius Communications Module is configured to transmit global data, then data from %G memory is sent to the GCM, GCM+, or GBC. The Series 90-20 and Micro output scans include discrete outputs only.

During the output scan, all Model 30 output modules are scanned in ascending reference address order.

If the CPU is in the **STOP** mode and the CPU is configured to not scan I/O during **STOP** mode, the output scan is skipped. The output scan is completed when all output data has been sent to all Model 30 output modules.

Logic Program Checksum Calculation

A checksum calculation is performed on the user program at the end of every sweep. Since it would take too long to calculate the checksum of the entire program, you can specify the number of words from 0 to 32 to be checksumed on the CPU detail screen.

If the calculated checksum does not match the reference checksum, the program checksum failure exception flag is raised. This causes a fault entry to be inserted into the PLC fault table and the PLC mode to be changed to **STOP**. If the checksum calculation fails, the programmer communications window is not affected. The default number of words to be checksummed is 8.

Programmer Communications Window

This part of the sweep is dedicated to communicating with the programmer. If there is a programmer attached, the CPU executes the programmer communications window. The programmer communications window will not execute if there is no programmer attached and no board to be configured in the system. Only one board is configured each sweep.

Support is provided for the Hand-Held Programmer and for other programmers that can connect to the serial port and use the Series Ninety Protocol (SNP) protocol. Support is also provided for programmer communications with intelligent option modules.

In the default limited window mode, the CPU performs one operation for the programmer each sweep, that is, it honors one service request or response to one key press. If the programmer makes a request that requires more milliseconds to process than the limit for the communications window (see Note), the request processing is spread out over several sweeps so that no sweep is impacted by more than the limit (see Note).

Note

The time limit for the communications window is as follows:

- 15 milliseconds for the 311 and 321 PLCs
- 8 milliseconds for the 313, 323 and 331 PLCs
- 10 milliseconds for the 340 and 341 PLCs
- 6 milliseconds for the 350 and higher PLCs

The following figure is a flow chart for the programmer communications portion of the sweep.

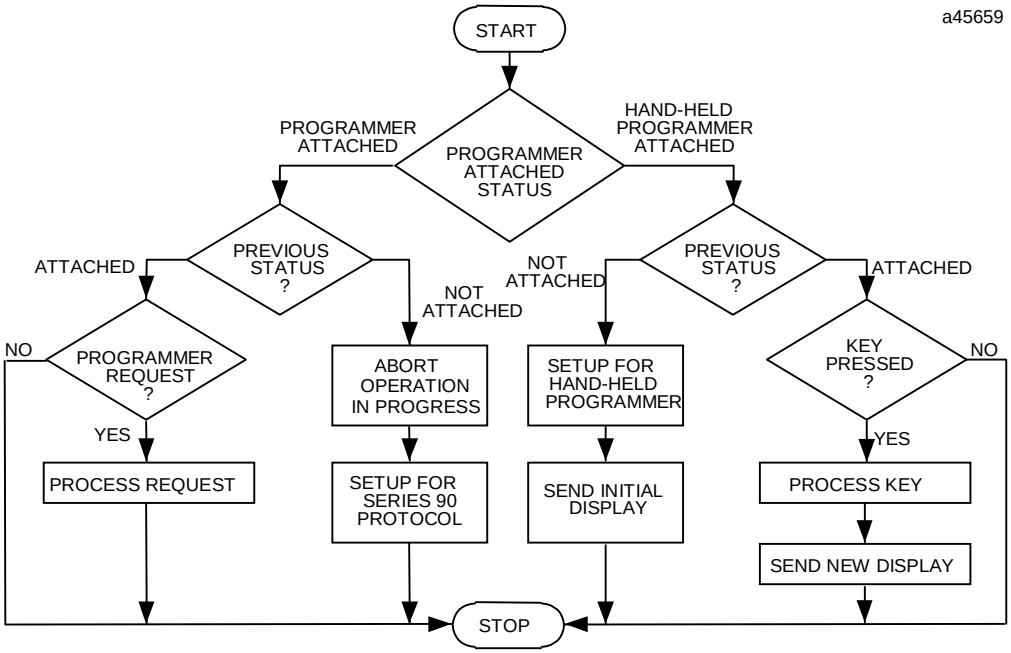


Figure 2-2. Programmer Communications Window Flow Chart

System Communications Window (Models 331 and Higher)

This is the part of the sweep where communications requests from intelligent option modules, such as the PCM, are processed (see flow chart). Requests are serviced on a first-come-first-served basis. However, since intelligent option modules are polled in a round-robin fashion, no intelligent option module has priority over any other intelligent option module.

In the default **Run-to-Completion** mode, the length of the system communications window is limited to 50 milliseconds. If an intelligent option module makes a request that requires more than 50 milliseconds to process, the request is spread out over multiple sweeps so that no one sweep is impacted by more than 50 milliseconds.

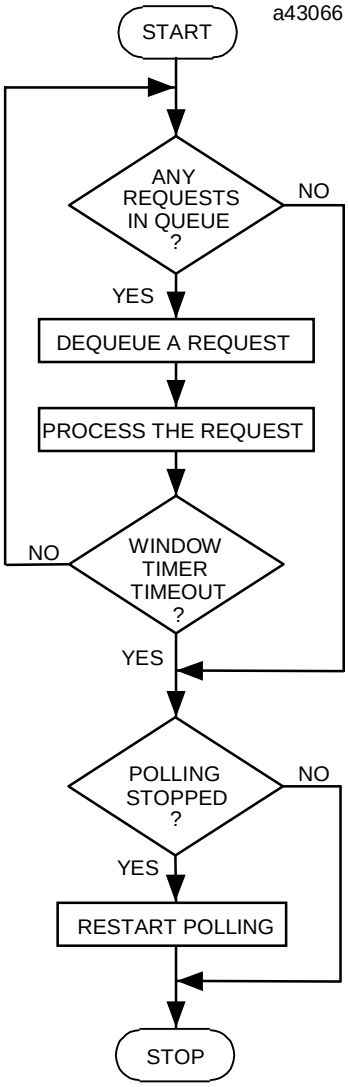


Figure 2-3. System Communications Window Flow Chart

PCM Communications with the PLC (Models 331 and Higher)

There is no way for intelligent option modules (IOM), such as the PCM, to interrupt the CPU when they need service. The CPU must poll each intelligent option module for service requests. This polling occurs asynchronously in the background during the sweep (see flow chart below).

When an intelligent option module is polled and sends the CPU a service request, the request is queued for processing during the system communications window.

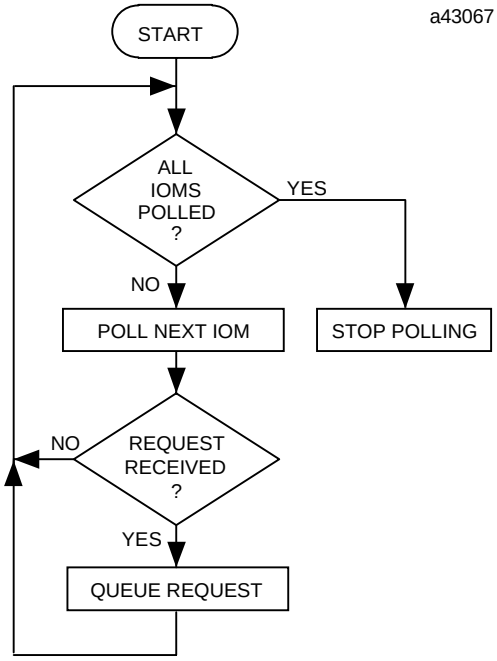


Figure 2-4. PCM Communications with the PLC

Standard Program Sweep Variations

In addition to the normal execution of the standard program sweep, certain variations can be encountered or forced. These variations, described in the following paragraphs, can be displayed and/or changed from the programming software.

Constant Sweep Time Mode

In the standard program sweep, each sweep executes as quickly as possible with a varying amount of time consumed each sweep. An alternative to this is **CONSTANT SWEEP TIME** mode, where each sweep consumes the same amount of time. You can achieve this by setting the Configured Constant Sweep, which will then become the default sweep mode, thereby taking effect each time the PLC goes from **STOP** to **RUN** mode. A value from 5 to 200 milliseconds (or up to 500 milliseconds for the 350 and 360 series PLC CPUs) for the constant sweep timer (default is 100 milliseconds) is supported.

Due to variations in the time required for various parts of the PLC sweep, the constant sweep time should be set at least 10 milliseconds higher than the sweep time that is displayed on the status line when the PLC is in **NORMAL SWEEP** mode. This prevents the occurrence of extraneous oversweep faults.

Use a constant sweep when I/O points or register values must be polled at a constant frequency, such as in control algorithms. One reason for using **CONSTANT SWEEP TIME** mode might be to ensure that I/O are updated at constant intervals. Another reason might be to ensure that a certain amount of time elapses between the output scan and the next sweep's input scan, permitting inputs to settle after receiving output data from the program.

If the constant sweep timer expires before the sweep completes, the entire sweep, including the windows, is completed. However, an oversweep fault is logged at the beginning of the next sweep.

Note

Unlike the Active Constant Sweep which can be edited only in **RUN** mode, the Configured Constant Sweep Mode can be edited only during **STOP** mode and you must "Store the configuration from the Programmer to the PLC" before the change will take effect. Once stored, this becomes the default sweep mode.

PLC Sweep When in STOP Mode

When the PLC is in **STOP** mode, the application program is not executed. Communications with the programmer and intelligent option modules continue. In addition, faulted board polling and board reconfiguration execution continue while in **STOP** mode. For efficiency, the operating system uses larger time-slice values than those used in **RUN** mode (usually about 50 milliseconds per window). You can choose whether or not the I/O is scanned. I/O scans may execute in **STOP** mode if the ***IOScan-Stop*** parameter on the CPU detail screen is set to **YES**.

Communication Window Modes

The default window mode for the programmer communication window is "Limited" mode. That means that if a request takes more than 6 milliseconds to process, it is processed over multiple sweeps, so that no one sweep is impacted by more than 6 milliseconds. For the 313, 323, and 331 CPUs, the sweep impact may be as much as 12 milliseconds during a **RUN**-mode store. The active window mode may be changed using the "Sweep Control" screen in Logicmaster—for instructions on changing the active window mode, refer to Chapter 5, "PLC Control and Status," in the *Logicmaster 90™ Series 90™-30/20/Micro Programming Software User's Manual* (GFK-0466)

Note

If the system window mode is changed to Limited, then option modules such as the PCM or GBC that communicate with the PLC using the system window will have less impact on sweep time, but response to their requests will be slower.

Key Switch on 350 and 360 Series CPUs: Change Mode and Flash Protect

Each of the 350 and 360 series CPUs has a key switch on the front of the module that allows you to protect Flash memory from being over-written. When you turn the key to the **ON**/**RUN** position, no one can change the Flash memory without turning the key to the OFF position.

Beginning with Release 7 of the 351and 352 CPUs, the same Key Switch has another function: it allows you to switch the PLC into **STOP** mode, into **RUN** mode, and to clear non-fatal faults as discussed in the next section.

Beginning with Release 8 of the 350 and the 360 series CPUs, the same Key Switch has an enhanced memory protection function: it can be used to provide two additional types of memory protection (see the "Using the Release 8 and Later Memory Protection" section).

Using the Release 7 and Later Key Switch

Unlike the Flash Protection capabilities in the earlier release, if you do not enable the Key Switch through the **RUN**/**STOP** Key Switch parameter in the CPU's configuration screen, the CPU does not have the enhanced control discussed here.

The operation of the Key Switch has the same safeguards and checks before the PLC goes to **RUN** mode just like the existing transition to **RUN** mode; that is, the PLC will not go to **RUN** mode via Key Switch input when the PLC is in **STOP**/**FAULT** mode. However, you can clear non-fatal faults and put the PLC in **RUN** mode through the use of the Key Switch.

If there are faults in the fault tables that ***are not fatal*** (that is, they do not cause the CPU to be placed in the **STOP**/**FAULT** mode), then the CPU will be placed in **RUN** mode the first time you turn the key from Stop to Run, and the fault tables will NOT be cleared.

If there are faults in the fault table that ***are fatal*** (CPU in **STOP**/**FAULT** mode), then the first transition of the Key Switch from the **STOP** position to the **RUN** position will cause the CPU **RUN** light to begin to flash at 2 Hz rate and a 5 second timer will begin. The flashing **RUN** light is an indication that there are fatal fault(s) in the fault tables. In which case, the CPU will NOT be placed in the **RUN** state even though the Key Switch is in **RUN** position.

Clearing the Fault Table with the Key Switch

If you turn the key from the **RUN** to **STOP** and back to **RUN** position during the 5 seconds when the **RUN** light is flashing this will cause the faults to be cleared and the CPU will be placed into **RUN** mode. The light will stop flashing and will go solid **ON** at this point. The switch is required to be kept in either **RUN** or **STOP** position for at least 1/2 second before switching back to the other position.

Note

If you allow the 5 second timer to expire (**RUN** light stops flashing) the CPU will remain in its original state, **STOP**/**FAULT** mode, with faults in the fault table. If you turn the Key Switch from the **STOP** to **RUN** position again at this time, the process will be repeated with this being the first transition.

The following table provides a summary of how the two CPU parameter settings affecting the Key Switch (R/S Switch and IOScan-Stop) and the Key Switch's physical position affect PLC.

R/S Key Switch Parameter in CPU Configuration	Key Switch Position	IOScan-Stop Parameter in CPU Configuration	PLC Operation
OFF	X	X	All PLC Programmer Modes are allowed.
ON	ON/RUN	X	All PLC Programmer Modes are allowed.
ON	OFF/STOP	X	PLC not allowed to go to RUN.
ON	Toggle Key Switch from OFF/STOP to ON/RUN	X	PLC goes to RUN if no fatal faults are present; otherwise, the RUN LED blinks for 5 seconds.
ON	Toggle Key Switch from ON/RUN to OFF/STOP	NO	PLC goes to STOP–NO IO
ON	Toggle Key Switch from ON/RUN to OFF/STOP	YES	PLC goes to STOP–IO

X=Has no affect regardless of setting

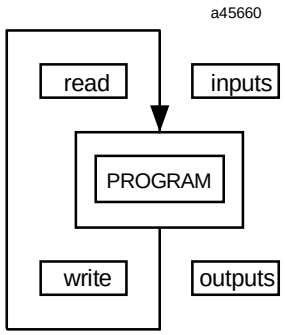
Enhanced Memory Protect with Release 8 and Later CPUs

In the Release 8 and later CPUs, the Key Switch has all the functionality discussed above, plus, by setting a parameter in the programming package, it can be used to protect RAM so that the RAM cannot be changed from the programming software. Two types of operations are blocked when this memory protection is enabled: the user program and configuration cannot be modified and the force and override of point data is not allowed. This is activated through the Mem Protect field in the 350 or 360 series CPUs module configuration screen in Logicmaster. The default is Disabled.

Section 2: Program Organization and User References/Data

The total logic size for the Series 90-30 programmable controller can be up to 6 KB in size for a Model 311 or Model 313 CPU, up to 16 KB in size for a Model 331, up to 32 KB in size for Model 340 CPUs, 74 KB for Model 350 (32 KB for program logic) and, prior to Release 9, up to 80 KB for Models 341, 351 and 352 CPUs. Beginning with Release 9 CPUs, some memory sizes for the 351, 352 and 360 series are configurable. (For detailed instructions and a discussion of memory sizes available, refer to the "Configurable Memory on 351 and higher CPUs" in Chapter 10, Section 3 of the *Logicmaster 90™ Series 90™-30/20/Micro Programming Software User's Manual* (GFK-0466K or later). A program for the Series 90-20 programmable controller can be up to 2 KB in size for a Model 211 CPU. A program for the Series 90 Micro programmable controller can be up to 6 KB in size, up to 12 KB for a 28-point Micro.

The user program contains logic that is used when it is started up. The maximum number of rungs allowed per logic block (main or subroutine) is 3000; for 90-30 PLCs, the maximum block size is 80 kilobytes for C blocks and 16 kilobytes for LD and SFC blocks, but in an SFC block some of the 16 KB is used for the internal data block. The logic is executed repeatedly by the PLC.



Refer to the *Series 90-30 Programmable Controller User's Manual*, GFK-0356, or the *Series 90-20 Programmable Controller User's Manual*, GFK-0551, for a listing of program sizes and reference limits for each model CPU.

All programs have a variable table that lists the variable and reference descriptions that have been assigned in the user program.

The block declaration editor lists subroutine blocks declared in the main program.

Subroutine Blocks (Series 90-30 PLC only)

A program can "call" subroutine blocks as it executes. A subroutine must be declared through the block declaration editor before a CALL instruction can be used for that subroutine. A maximum of 64 subroutine block declarations in the program and 64 CALL instructions are allowed for each logic block in the program. The maximum size of a subroutine block is 16 KB or 3000 rungs, but the main program and all subroutines must fit within the logic size constraints for that CPU model.

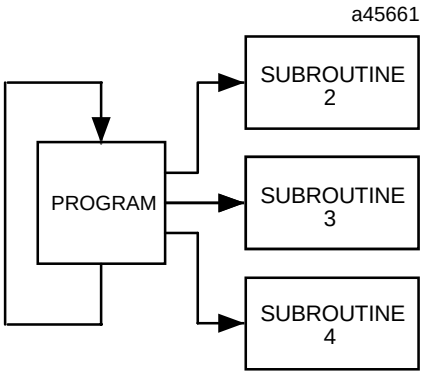
Note

Subroutine blocks are not available for the Series 90-20 PLC or for the Micro.

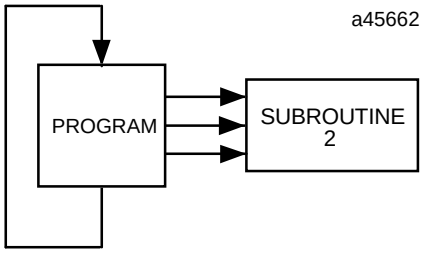
The use of subroutines is optional. Dividing a program into smaller subroutines can simplify programming, enhance understanding of the control algorithm, and reduce the overall amount of logic needed for the program.

Examples of Using Subroutine Blocks

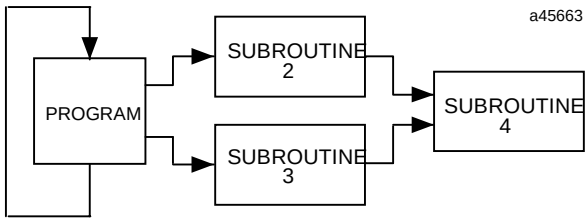
As an example, the logic for a program could be divided into three subroutines, each of which could be called as needed from the program. In this example, the program block might contain little logic, serving primarily to sequence the subroutine blocks.



A subroutine block can be used many times as the program executes. Logic which needs to be repeated several times in a program could be entered in a subroutine block. Calls would then be made to that subroutine block to access the logic. In this way, total program size is reduced.



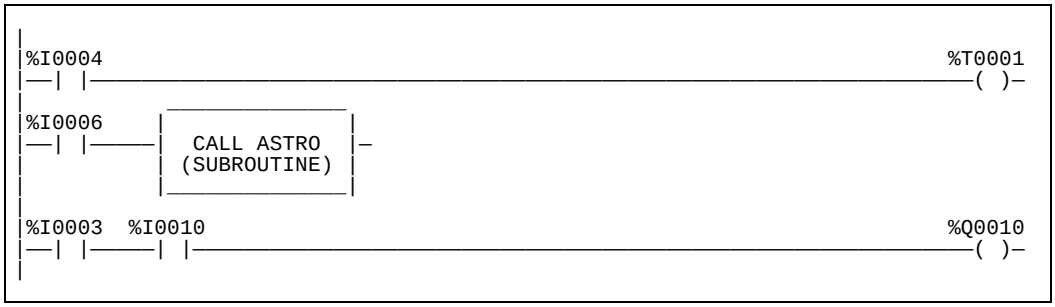
In addition to being called from the program, subroutine blocks can also be called by other subroutine blocks. A subroutine block may even call itself.



The PLC will only allow eight nested calls before an "Application Stack Overflow" fault is logged and the PLC transitions to **`STOP/Fault`** mode. The call level nesting counts the main program as level 1.

How Blocks Are Called

A subroutine block executes when called from the program logic in the program or from another block.



This example shows the subroutine CALL instruction as it will appear in the calling block.

Periodic Subroutines

Version 4.20 or later of the 340 and higher CPUs support periodic subroutines. Please note the following restrictions:

1. Timer (TMR, ONDTR, and OFDTR) function blocks will not execute properly within a periodic subroutine. A DOIO function block within a periodic subroutine whose reference range includes references assigned to a Smart I/O Module (HSC, Power Mate APM, Genius, etc.) will cause the CPU to lose communication with the module. The FST_SCN and LST_SCN contacts (%S1 and %S2) will have an indeterminate value during execution of the periodic subroutine. A periodic subroutine cannot call or be called by other subroutines.
2. The latency for the periodic subroutine (that is, the maximum interval between the time the periodic subroutine should have executed and the time it actually executes) can be around .35 milliseconds if there is no PCM, CMM, or ADC module in the main rack. If there is a PCM, CMM or ADC module in the main rack—even if it is not configured or used—the latency can be almost 2.25 milliseconds. For that reason, use of the periodic subroutine with PCM-based products is ***not*** recommended.

User References

The data used in an application program is stored as either register or discrete references.

Table 2-3. Register References

Type	Description
%R	The prefix %R is used to assign system register references, which will store program data such as the results of calculations.
%AI	The prefix %AI represents an analog input register. This prefix is followed by the register address of the reference (for example, %AI0015). An analog input register holds the value of one analog input or other value.
%AQ	The prefix %AQ represents an analog output register. This prefix is followed by the register address of the reference (for example, %AQ0056). An analog output register holds the value of one analog output or other value.

Note

All register references are retained across a power cycle to the CPU.

Table 2-4. Discrete References

Type	Description
%I	The %I prefix represents input references. This prefix is followed by the reference's address in the input table (for example, %I00121). %I references are located in the input status table, which stores the state of all inputs received from input modules during the last input scan. A reference address is assigned to discrete input modules using the configuration software or the Hand-Held Programmer. Until a reference address is assigned, no data will be received from the module. %I data can be retentive or non-retentive.
%Q	The %Q prefix represents output references. The coil check function of Logicmaster 90-30/20/Micro software checks for multiple uses of %Q references with relay coils or outputs on functions. Beginning with Release 3 of the software, you can select the level of coil checking desired (`SINGLE`, `WARN MULTIPLE`, or `MULTIPLE`). Refer to the *Logicmaster 90-30/20Micro Programming Software User's Manual*, GFK-0466, for more information about this feature. The %Q prefix is followed by the reference's address in the output table (for example, %Q00016). %Q references are located in the output status table, which stores the state of the output references as last set by the application program. This output status table's values are sent to output modules during the output scan. A reference address is assigned to discrete output modules using the configuration software or the Hand-Held Programmer. Until a reference address is assigned, no data is sent to the module. A particular %Q reference may be either retentive or non-retentive. *
%M	The %M prefix represents internal references. The coil check function checks for multiple uses of %M references with relay coils or outputs on functions. Beginning with Release 3 of the software, you can select the level of coil checking desired (`SINGLE`, `WARN MULTIPLE`, or `MULTIPLE`). Refer to the *Logicmaster 90-30/20Micro Programming Software User's Manual*, GFK-0466, for more information about this feature. A particular %M reference may be either retentive or non-retentive. *

* Retentiveness is based on the type of coil. For more information, refer to "Retentiveness of Data" on page 2-21.

Table 2-4. Discrete References - Continued

Type	Description
%T	The %T prefix represents temporary references. These references are never checked for multiple coil use and can, therefore, be used many times in the same program even when coil use checking is enabled. %T may be used to prevent coil use conflicts while using the cut/paste and file write/include functions. Because this memory is intended for temporary use, it is never retained through power loss or **RUN-TO-STOP-TO-RUN** transitions and cannot be used with retentive coils.
%S	The %S prefix represents system status references. These references are used to access special PLC data, such as timers, scan information, and fault information. System references include %S, %SA, %SB, and %SC references. %S, %SA, %SB, and %SC can be used on any contacts. %SA, %SB, and %SC can be used on retentive coils –(M)–. %S can be used as word or bit-string input arguments to functions or function blocks. %SA, %SB, and %SC can be used as word or bit-string input or output arguments to functions and function blocks.
%G	The %G prefix represents global data references. These references are used to access data shared among several PLCs. %G references can be used on contacts and retentive coils because %G memory is always retentive. %G cannot be used on non-retentive coils.

Transitions and Overrides

The %I, %Q, %M, and %G user references have associated transition and override bits. %T, %S, %SA, %SB, and %SC references have transition bits, but not override bits. The CPU uses transition bits for counters and transitional coils. Note that counters do not use the same kind of transition bits as coils. Transition bits for counters are stored within the locating reference.

In the Model 331 and higher CPUs, override bits can be set. When override bits are set, the associated references cannot be changed from the program or the input device; they can only be changed on command from the programmer. CPU Models 323, 321, 313 311, 211, and the Micro CPUs do not support overriding discrete references.

Retentiveness of Data

Data is said to be retentive if it is saved by the PLC when the PLC is stopped. The Series 90 PLC preserves program logic, fault tables and diagnostics, overrides and output forces, word data (%R, %AI, %AQ), bit data (%I, %SC, %G, fault bits and reserved bits), %Q and %M data (unless used with non-retentive coils), and word data stored in %Q and %M. %T data is not saved. Although, as stated above, %SC bit data is retentive, the defaults for %S, %SA, and %SB are non-retentive.

%Q and %M references are non-retentive (that is, cleared at power-up when the PLC switches from **STOP** to **RUN**) whenever they are used with non-retentive coils. Non-retentive coils include coils —( )—, negated coils —(/)—, SET coils —(S)—, and RESET coils —(R)—.

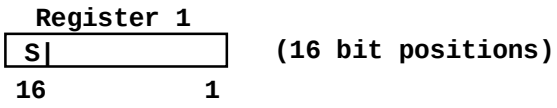
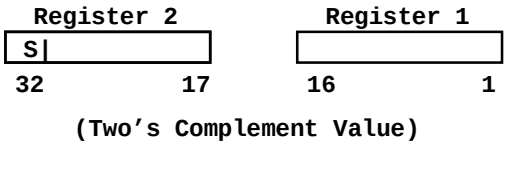
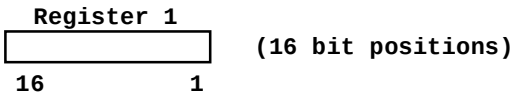
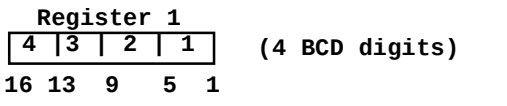
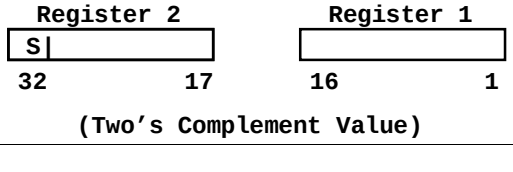
When %Q or %M references are used with retentive coils, or are used as function block outputs, the contents are retained through power loss and **RUN-TO-STOP-TO-RUN** transitions. Retentive coils include retentive coils —(M)—, negated retentive coils —(/M)—, retentive SET coils —(SM)—, and retentive RESET coils —(RM)—.

The last time a %Q or %M reference is programmed on a coil instruction determines whether the %Q or %M reference is retentive or non-retentive based on the coil type. For example, if %Q0001 was last programmed as the reference of a retentive coil, the %Q0001 data will be retentive. However, if %Q0001 was last programmed on a non-retentive coil, then the %Q0001 data will be non-retentive.

Data Types

Data types include the following:

Table 2-5. Data Types

Type	Name	Description	Data Format
INT	Signed Integer	Signed integers use 16-bit memory data locations, and are represented in 2's complement notation. The valid range of an INT data type is –32,768 to +32,767.	
DINT	Double Precision Signed Integer	Double precision signed integers are stored in 32-bit data memory locations (actually two consecutive 16-bit memory locations) and represented in 2's complement notation. (Bit 32 is the sign bit.) The valid range of a DINT data type is –2,147,483,648 to +2,147,483,867.	
BIT	Bit	A Bit data type is the smallest unit of memory. It has two states, 1 or 0. A BIT string may have length N.	
BYTE	Byte	A Byte data type has an 8-bit value. The valid range is 0 to 255 (0 to FF in hexadecimal).	
WORD		A Word data type uses 16 consecutive bits of data memory; but, instead of the bits in the data location representing a number, the bits are independent of each other. Each bit represents its own binary state (1 or 0), and the bits are not looked at together to represent an integer number. The valid range of word values is 0 to FFFF.	
BCD-4	Four-Digit Binary Coded Decimal	Four-digit BCD numbers use 16-bit data memory locations. Each BCD digit uses four bits and can represent numbers between 0 and 9. This BCD coding of the 16 bits has a legal value range of 0 to 9999.	
REAL	Floating Point	Real numbers use 32 consecutive bits (actually two consecutive 16-bit memory locations). The range of numbers that can be stored in this format is from ± 1.401298E-45 to ± 3.402823E+38.	

S = Sign bit (0 = positive, 1 = negative).

System Status References

System status references in the Series 90 PLC are assigned to %S, %SA, %SB, and %SC memory. They each have a nickname. Examples of time tick references include T_10MS, T_100MS, T_SEC, and T_MIN. Examples of convenience references include FST_SCN, ALW_ON, and ALW_OFF.

Note

%S bits are read-only bits; do not write to these bits. You may, however, write to %SA, %SB, and %SC bits.

Listed below are available system status references, which may be used in an application program. When entering logic, either the reference or the nickname can be used. Refer to chapter 3, "Fault Explanations and Correction," for more detailed fault descriptions and information on correcting the fault.

You cannot use these special names in another context.

Table 2-6. System Status References

Reference	Nickname	Definition
%S0001	FST_SCN	Set to 1 when the current sweep is the first sweep.
%S0002	LST_SCN	Reset from 1 to 0 when the current sweep is the last sweep.
%S0003	T_10MS	0.01 second timer contact.
%S0004	T_100MS	0.1 second timer contact.
%S0005	T_SEC	1.0 second timer contact.
%S0006	T_MIN	1.0 minute timer contact.
%S0007	ALW_ON	Always ON.
%S0008	ALW_OFF	Always OFF.
%S0009	SY_FULL	Set when the PLC fault table fills up. Cleared when an entry is removed from the PLC fault table and when the PLC fault table is cleared.
%S0010	IO_FULL	Set when the I/O fault table fills up. Cleared when an entry is removed from the I/O fault table and when the I/O fault table is cleared.
%S0011	OVR_PRE	Set when an override exists in %I, %Q, %M, or %G memory.
%S0013	PRG_CHK	Set when background program check is active.
%S0014	PLC_BAT	Set to indicate a bad battery in a Release 4 or later CPU. The contact reference is updated once per sweep.

Table 2-6. System Status References - Continued

Reference	Name	Definition
%S0017	SNPXACT	SNP-X host is actively attached to the CPU.
%S0018	SNPX_RD	SNP-X host has read data from the CPU.
%S0019	SNPX_WT	SNP-X host has written data to the CPU.
%S0020		Set ON when a relational function using REAL data executes successfully. It is cleared when either input is NaN (Not a Number).
%S0032		Reserved for use by the programming software.
%SA0001	PB_SUM	Set when a checksum calculated on the application program does not match the reference checksum. If the fault was due to a temporary failure, the discrete bit can be cleared by again storing the program to the CPU. If the fault was due to a hard RAM failure, the CPU must be replaced.
%SA0002	OV_SWP	Set when the PLC detects that the previous sweep took longer than the time specified by the user. Cleared when the PLC detects that the previous sweep did not take longer than the specified time. It is also cleared during the transition from **STOP** to **RUN** mode. Only valid if the PLC is in **CONSTANT SWEEP** mode.
%SA0003	APL_FLT	Set when an application fault occurs. Cleared when the PLC transitions from **STOP** to **RUN** mode.
%SA0009	CFG_MM	Set when a configuration mismatch is detected during system power-up or during a store of the configuration. Cleared by powering up the PLC when no mismatches are present or during a store of configuration that matches hardware.
%SA0010	HRD_CPU	Set when the diagnostics detects a problem with the CPU hardware. Cleared by replacing the CPU module.
%SA0011	LOW_BAT	Set when a low battery fault occurs. Cleared by replacing the battery and ensuring that the PLC powers up without the low battery condition.
%SA0014	LOS_IOM	Set when an I/O module stops communicating with the PLC CPU. Cleared by replacing the module and cycling power on the main rack.
%SA0015	LOS_SIO	Set when an option module stops communicating with the PLC CPU. Cleared by replacing the module and cycling power on the main rack.
%SA0019	ADD_IOM	Set when an I/O module is added to a rack. Cleared by cycling power on the main rack and when the configuration matches the hardware after a store.
%SA0020	ADD_SIO	Set when an option module is added to a rack. Cleared by cycling power on the main rack and when the configuration matches the hardware after a store.
%SA0027	HRD_SIO	Set when a hardware failure is detected in an option module. Cleared by replacing the module and cycling power on the main rack.
%SA0031	SFT_SIO	Set when an unrecoverable software fault is detected in an option module. Cleared by cycling power on the main rack and when the configuration matches the hardware.

Table 2-6. System Status References - Continued

Reference	Nickname	Definition
%SB0010	BAD_RAM	Set when the CPU detects corrupted RAM memory at power-up. Cleared when the CPU detects that RAM memory is valid at power-up.
%SB0011	BAD_PWD	Set when a password access violation occurs. Cleared when the PLC fault table is cleared.
%SB0013	SFT_CPU	Set when the CPU detects an unrecoverable error in the software. Cleared by clearing the PLC fault table.
%SB0014	STOR_ER	Set when an error occurs during a programmer store operation. Cleared when a store operation is completed successfully.
%SC0009	ANY_FLT	Set when any fault occurs. Cleared when both fault tables have no entries.
%SC0010	SY_FLT	Set when any fault occurs that causes an entry to be placed in the PLC fault table. Cleared when the PLC fault table has no entries.
%SC0011	IO_FLT	Set when any fault occurs that causes an entry to be placed in the I/O fault table. Cleared when the I/O fault table has no entries.
%SC0012	SY_PRES	Set as long as there is at least one entry in the PLC fault table. Cleared when the PLC fault table has no entries.
%SC0013	IO_PRES	Set as long as there is at least one entry in the I/O fault table. Cleared when the I/O fault table has no entries.
%SC0014	HRD_FLT	Set when a hardware fault occurs. Cleared when both fault tables have no entries.
%SC0015	SFT_FLT	Set when a software fault occurs. Cleared when both fault tables have no entries.

Note: Any %S reference not listed here is reserved and not to be used in program logic.

Function Block Structure

Each rung of logic is composed of one or more programming instructions. These may be simple relays or more complex functions.

Format of Ladder Logic Relays

The programming software includes several types of relay functions. These functions provide basic flow and control of logic in the program. Examples include a normally open relay contact and a negated coil. Each of these relay contacts and coils has one input and one output. Together, they provide logic flow through the contact or coil.

Each relay contact or coil must be given a reference which is entered when selecting the relay. For a contact, the reference represents a location in memory that determines the flow of power into the contact. In the following example, if reference %I0122 is ON, power will flow through this relay contact.

```
%I0122
-| |-
```

For a coil, the reference represents a location in memory that is controlled by the flow of power into the coil. In this example, if power flows into the left side of the coil, reference %Q0004 is turned ON.

```
%Q0004
-( )-
```

The programming software and the Hand-Held Programmer both have a coil check function that checks for multiple uses of %Q or %M references with relay coils or outputs on functions.

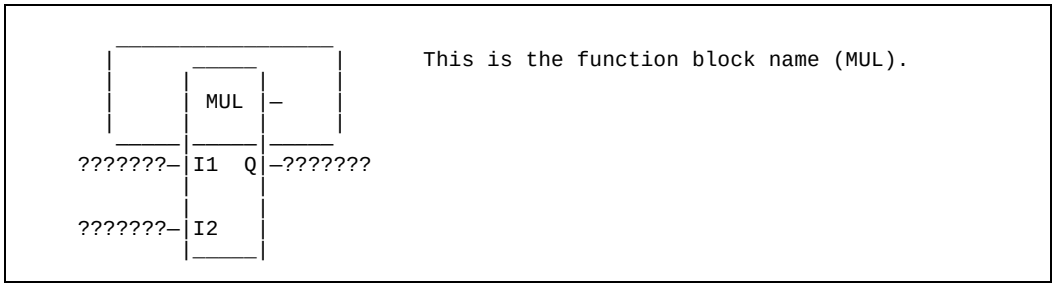
Format of Program Function Blocks

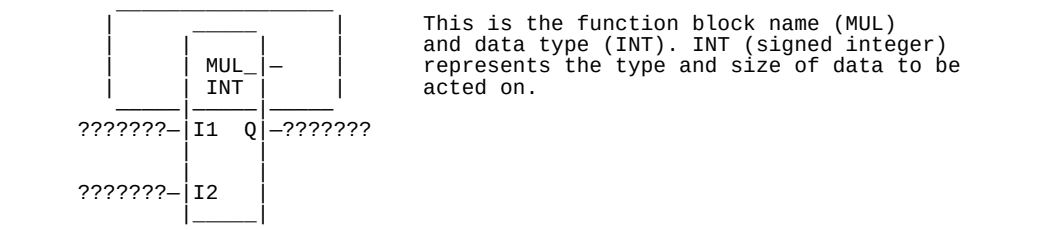
Some functions are very simple, like the MCR function, which is shown with the abbreviated name of the function within brackets:

```
-[ MCR ]-
```

Other functions are more complex. They may have several places where you will enter information to be used by the function.

The generic function block illustrated below is multiplication (MUL); parameters vary with the type of function block. Its parts are typical of many program functions. The upper part of the function block shows the name of the function. It may also show a data type; in this case, signed integer.



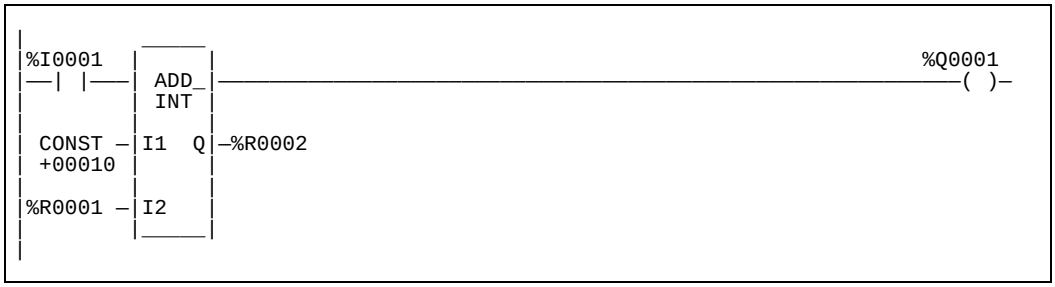


Many program functions allow you to select the data type for the function after selecting the function. For example, the data type for the MUL function could be changed to double precision signed integer. Additional information on data types is provided earlier in this chapter.

Function Block Parameters

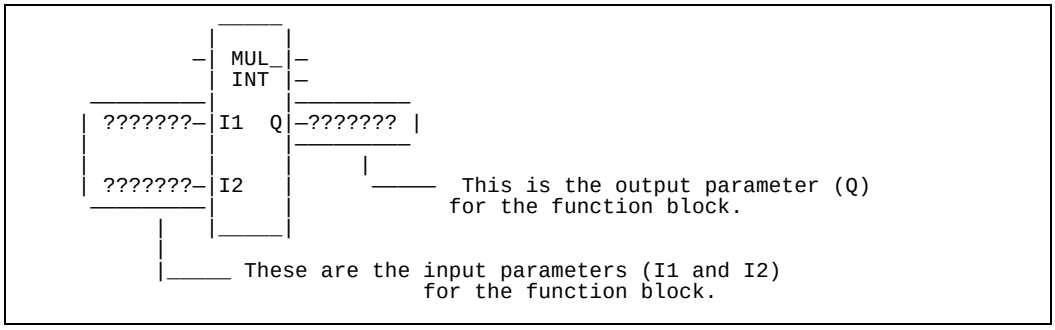
Each line entering the left side of a function block represents an input for that function. There are two forms of input that can be passed into a function block: constants and references. A constant is an explicit value. A reference is the address of a value.

In the following example, input parameter I1 comes into the ADD function block as a constant, and input parameter I2 comes in as a reference.



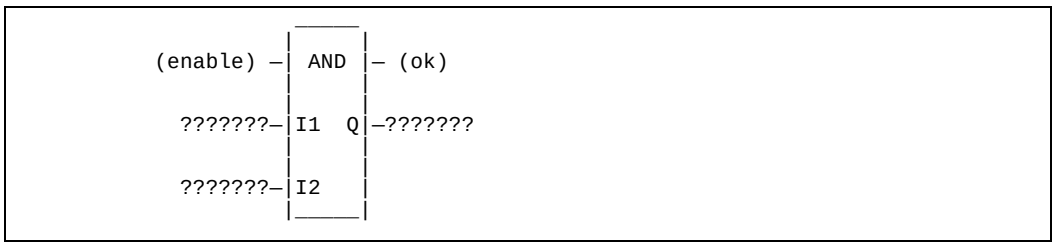
Each line exiting the right side of the function block represents an output. There is only one form of output from a function block or reference. Outputs can never be written to constants.

Where the question marks appear on the left of a function block, you will enter either the data itself, a reference location where the data is found, or a variable representing the reference location where the data is found. Where question marks appear on the right of a function block, you will usually enter a reference location for data to be output by the function block or a variable that represents the reference location for data to be output by the function block.

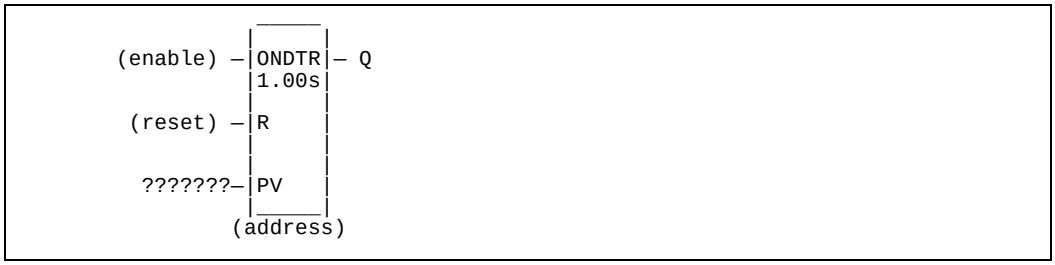


Most function blocks do not change input data; instead, they place the result of the operation in an output reference.

For functions which operate on tables, a length can be selected for the function. In the following function block, a string length of up to 256 words can be selected for the logical AND function.

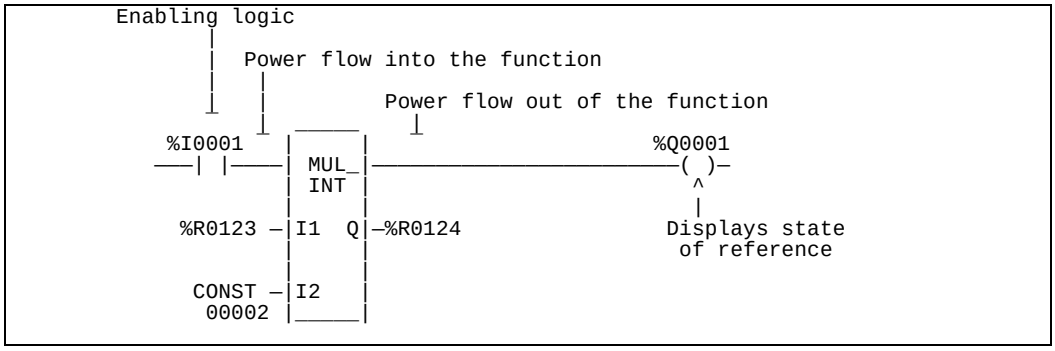


Timer, counter, BITSEQ, and ID functions require an address for the location of three words (registers) which store the current value, preset value, and a control word or “Instance” of the function.



Power Flow In and Out of a Function

Power flows into a function block on the upper left. Often, enabling logic is used to control power flow to a function block; otherwise, the function block executes unconditionally each CPU sweep.



Note

Function blocks cannot be tied directly to the left power rail. You can use %S7, the ALW_ON (always on) bit with a normally open contact tied to the power rail to call a function every sweep.

Power flows out of the function block on the upper right. It may be passed to other program logic or to a coil (optional). Function blocks pass power when they execute successfully.

Section 3: Power-Up and Power-Down Sequences

There are two possible power-up sequences in the Series 90-30 PLC; a cold power-up and a warm power-up. The CPU normally uses the cold power-up sequence. However, in a Model 331 or higher PLC system, if the time that elapses between a power-down and the next power-up is less than five seconds, the warm power-up sequence is used.

Power-Up

A cold power-up consists of the following sequence of events. A warm power-up sequence skips Step 1.

1. The CPU will run diagnostics on itself. This includes checking a portion of battery-backed RAM to determine whether or not the RAM contains valid data.
2. If an EPROM, EEPROM, or flash is present and the PROM power-up option in the PROM specifies that the PROM contents should be used, the contents of PROM are copied into RAM memory. If an EPROM, EEPROM, or flash is not present, RAM memory remains the same and is not overwritten with the contents of PROM.
3. The CPU interrogates each slot in the system to determine which boards are present.
4. The hardware configuration is compared with software configuration to ensure that they are the same. Any mismatches detected are considered faults and are alarmed. Also, if a board is specified in the software configuration but a different module is present in the actual hardware configuration, this condition is a fault and is alarmed.
5. If there is no software configuration, the CPU will use the default configuration.
6. The CPU establishes the communications channel between itself and any intelligent modules.
7. In the final step of the execution, the mode of the first sweep is determined based on CPU configuration. If **RUN** mode, the sweep proceeds as described under "**STOP**-to-**RUN** Mode Transition." Figure 2-5 on the next page shows the decision sequence for the CPU when it decides whether to copy from PROM or to power-up in **STOP** or **RUN** mode.

Note

Steps 2 through 6 above do not apply to the Series 90 Micro PLC. For information about the power-up and power-down sequences for the Micro, refer to the "Power-up and Power-down Sequences" section of Chapter 5, "System Operation," in the *Series 90 Micro PLC User's Manual* (GFK-1065).

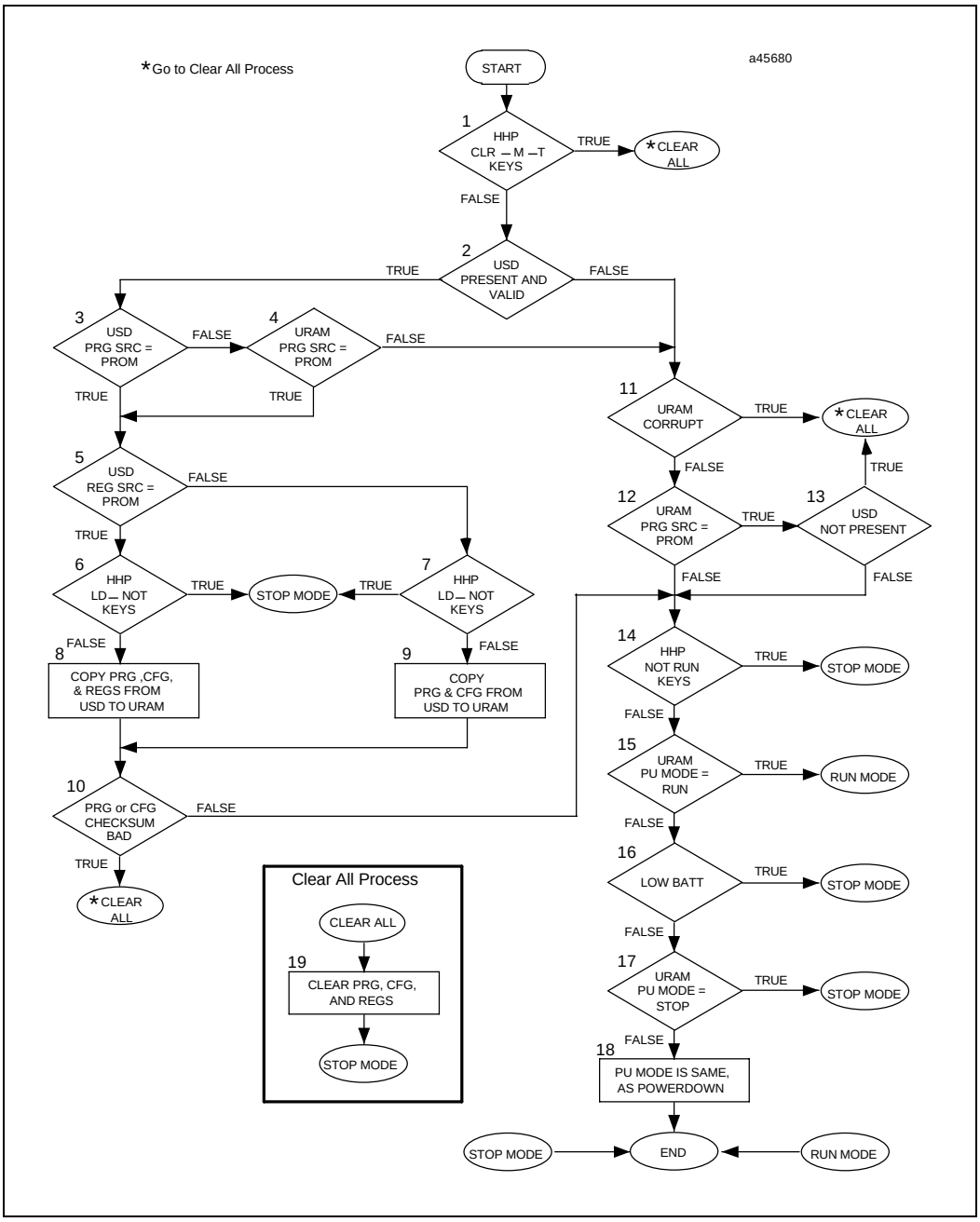


Figure 2-5. Power-Up Sequence

Prior to the START statement on the Power Up Flowchart, the CPU goes through power up diagnostics which test various peripheral devices used by the CPU and tests RAM. After completing diagnostics, internal data structures and peripheral devices used by the CPU get initialized. The CPU then determines if User Ram has been corrupted. If User Ram is corrupted the user program and configuration are cleared out and defaulted and all user registers are cleared.

FLOW CHART TERMS:
PRG = user program
CFG = user configuration
REGS = user registers (%I, %Q, %M, %G, %R, %AI, and %AQ references).
USD = user storage device, either an EEPROM or flash device.
URAM = non-volatile user ram which contains PRG, CFG, and REGS.

FLOW CHART EXPANDED TEXT:

(1) Are the <CLR> and <M_T> keys being pressed on the HHP during power-up to clear all URAM?

(2) Is the USD present (could only be missing on models that use EEPROM device) and is the information on the USD valid?

(3) Is the PRG SRC parameter in the USD set to Prom meaning to load the PRG and CFG from the USD device?

(4) Is the PRG SRC parameter in the URAM set to Prom meaning to load the PRG and CFG from the USD device?

(5) Is the REG SRC parameter in the USD set to Prom meaning to load the REGS from the USD device?

(6 & 7) Are the <LD> and <NOT> keys being pressed on the HHP during power-up to keep the PRG, CFG, and REGS from being loaded from USD?

(8) Copy PRG, CFG, and REGS from the USD to URAM.

(9) COPY PRG, and CFG from the USD to URAM.

(10) Is the PRG or CFG checksums just loaded from USD invalid?

(11) Is the URAM corrupted? Could be due to being powered down with out a battery attached or a low battery. Could also be due to updating firmware.

(12) Is the PRG SRC parameter in the URAM set to Prom meaning to load the PRG and CFG from the USD device?

(13) Is the USD present? Only applicable to models that use EEPROM device.

(14) Are the <NOT> and <**RUN**> keys being pressed on the HHP during power-up to unconditionally power-up in Stop Mode?

(15) Is the PWR UP parameter in URAM set to **RUN**?

(16) Is the battery low?

(17) Is the PWR UP parameter in URAM set to **STOP**?

(18) Set the power up mode to what ever the power down mode was.

(19) Clear PRG, CFG, and REGS.

Note

The first part of this chart on the previous page does not apply to the Series 90 Micro PLC. For information about the power-up and power-down sequences for the Micro, refer to the “Power-up and Power-down Sequences” section of Chapter 5, “System Operation,” in the *Series 90 Micro PLC User’s Manual* (GFK-1065).

Power-Down

System power-down occurs when the power supply detects that incoming AC power has dropped for more than one power cycle or the output of the 5-volt power supply has fallen to less than 4.9 volts DC.

Section 4: Clocks and Timers

Clocks and timers provided by the Series 90-30 PLC include an elapsed time clock, a time-of-day clock (Models 331, 340/341, 350 and 360 series of 90-30 CPUs and the 28-point Micro), a watchdog timer, and a constant sweep timer. Three types of timer function blocks include an on-delay timer, an off-delay timer, and a retentive on-delay timer (also called a watch clock timer). Four time-tick contacts cycle on and off for 0.01 second, 0.1 second, 1.0 second, and 1 minute intervals.

Elapsed Time Clock

The elapsed time clock uses 100 microsecond "ticks" to track the time elapsed since the CPU powered on. The clock is not retentive across a power failure; it restarts on each power-up. Once per second the hardware interrupts the CPU to enable a seconds count to be recorded. This seconds count rolls over approximately 100 years after the clock begins timing.

Because the elapsed time clock provides the base for system software operations and timer function blocks, it can not be reset from the user program or the programmer. However, the application program can read the current value of the elapsed time clock by using Service Request 16. Elapsed power down, reported by use of Service Request 29, also utilizes this clock.

Time-of-Day Clock

The time of day in the 28-point Micro and Series 90-30 PLC Model 331 and higher is maintained by a hardware time-of-day clock. The time-of-day clock maintains seven time functions:

- Year (two digits)
- Month
- Day of month
- Hour
- Minute
- Second
- Day of week

The time-of-day clock is battery-backed and maintains its present state across a power failure. However, unless you initialize the clock, the values it contains are meaningless. The application program can read and set the time-of-day clock using Service Request #7. The time-of-day clock can also be read and set from the CPU configuration software.

The time-of-day clock is designed to handle month-to-month and year-to-year transitions. It automatically compensates for leap years until the year 2079.

Watchdog Timer

A watchdog timer in the Series 90-30 PLC is designed to catch catastrophic failure conditions that result in an unusually long sweep. The timer value for the watchdog timer is 200 milliseconds (500 milliseconds in the 350 and 360 series of PLC CPUs); this is a fixed value that cannot be changed. The watchdog timer always starts from zero at the beginning of each sweep.

For 331 and lower model 90-30 CPUs, if the watchdog timeout value is exceeded, the OK LED goes off; the CPU is placed in reset and completely shuts down; and outputs go to their default state. No communication of any form is possible, and all microprocessors on all boards are halted. To recover, power must be cycled on the rack containing the CPU. In the 90-20, Series 90 Micro and 340 and higher 90-30 CPUs, A watchdog timeout causes the CPU to reset, execute its powerup logic, generate a watchdog failure fault, and change its mode to **STOP**.

Constant Sweep Timer

The constant sweep timer controls the length of a program sweep when the Series 90-30 PLC operates in **CONSTANT SWEEP TIME** mode. In this mode of operation, each sweep consumes the same amount of time. Typically, for most application programs, the input scan, application program logic scan, and output scan do not require exactly the same amount of execution time in each sweep. The value of the constant sweep timer is set by the programmer and can be any value from 5 to the value of the watchdog timer (default is 100 milliseconds).

If the constant sweep timer expires before the completion of the sweep and the previous sweep was not oversweep, the PLC places an oversweep alarm in the PLC fault table. At the beginning of the next sweep, the PLC sets the OV_SWP fault contact. The OV_SWP contact is reset when the PLC is not in **CONSTANT SWEEP TIME** mode or the time of the last sweep did not exceed the constant sweep timer.

Time-Tick Contacts

The Series 90 PLC provides four time-tick contacts with time durations of 0.01 second, 0.1 second, 1.0 second, and 1 minute. The state of these contacts does not change during the execution of the sweep. These contacts provide a pulse having an equal on and off time duration. The contacts are referenced as T_10MS (0.01 second), T_100MS (0.1 second), T_SEC (1.0 second), and T_MIN (1 minute).

The following timing diagram represents the on/off time duration of these contacts.

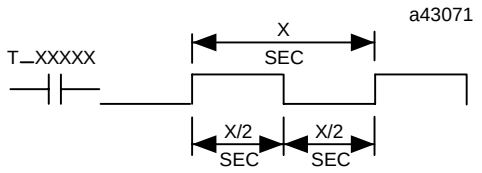


Figure 2-6. Time-Tick Contact Timing Diagram

Section 5: System Security

Security in Series 90-30, Series 90-20, and in the Micro PLCs is designed to prevent unauthorized changes to the contents of a PLC. There are four security levels available in the PLC. The first level, which is always available, provides only the ability to read PLC data; no changes are permitted to the application. The other three levels have access to each level protected by a password.

Each higher privilege level permits greater change capabilities than the lower level(s). Privilege levels accumulate in that the privileges granted at one level are a combination of that level, plus all lower levels. The levels and their privileges are:

Privilege Level	Description
Level 1	Any data, except passwords may be read. This includes all data memories (%I, %Q, %AQ, %R, etc.), fault tables, and all program block types (data, value, and constant). No values may be changed in the PLC.
Level 2	This level allows write access to the data memories (%I, %R, etc.).
Level 3	This level allows write access to the application program in **STOP** mode only.
Level 4	This is the default level for systems which have no passwords set. The default level for a system with passwords is to the highest unprotected level. This level, the highest, allows read and write access to all memories as well as passwords in both **RUN** and **STOP** mode. (Configuration data cannot be changed in **RUN** mode.)

Passwords

There is one password for each privilege level in the PLC. (No password can be set for level 1 access.) Each password may be unique; however, the same password can be used for more than one level. To maintain compatibility with the Hand-Held Programmer, passwords should be up to four Hex characters in length (up to 7 accepted in the programming software); they can only be entered or changed with the programming software or the Hand-Held Programmer.

A privilege level change is in effect only as long as communications between the PLC and the programmer are intact. There does not need to be any activity, but the communications link must not be broken. If there is no communication for 15 minutes, the privilege level returns to the highest unprotected level.

Upon connection of the PLC, the programming software requests the protection status of each privilege level from the PLC. The programming software then requests the PLC to move to the highest unprotected level, thereby giving the programming software access to the highest unprotected level without having to request any particular level. When the Hand-Held Programmer is connected to the PLC, the PLC reverts to the highest unprotected level.

Privilege Level Change Requests

A programmer requests a privilege level change by supplying the new privilege level and the password for that level. A privilege level change is denied if the password sent by the programmer does not agree with the password stored in the PLC's password access table for the requested level. The current privilege level is maintained and no change will occur. If you attempt to access or modify information in the PLC using the Hand-Held Programmer without the proper privilege level, the Hand-Held Programmer will respond with an error message that the access is denied.

Locking/Unlocking Subroutines

Subroutine blocks can be locked and unlocked using the block locking feature of programming software. Two types of locks are available:

Type of Lock	Description
View	Once locked, you cannot zoom into that subroutine.
Edit	Once locked, the information in the subroutine cannot be edited.

A previously view locked or edit locked subroutine may be unlocked in the block declaration editor unless it is permanently view locked or permanently edit locked.

A search or search and replace function may be performed on a view locked subroutine. If the target of the search is found in a view locked subroutine, one of the following messages is displayed, instead of logic:

```
Found in locked block <block_name>   (Continue/Quit)
```

or

```
Cannot write to locked block <block_name>  (Continue/Quit)
```

You may continue or abort the search.

Folders that contain locked subroutines may be cleared or deleted. If a folder contains locked subroutines, these blocks remain locked when the programming software Copy, Backup, and Restore folder functions are used.

Permanently Locking a Subroutine

In addition to VIEW LOCK and EDIT LOCK, there are two types of permanent locks. If a PERMANENT VIEW LOCK is set, all zooms into a subroutine are denied. If a PERMANENT EDIT LOCK is set, all attempts to edit the block are denied.

Caution

The permanent locks differ from the regular VIEW LOCK and EDIT LOCK in that once set, they cannot be removed.

Once a PERMANENT EDIT LOCK is set, it can only be changed to a PERMANENT VIEW LOCK. A PERMANENT VIEW LOCK cannot be changed to any other type of lock.

Section 6: Series 90-30, 90-20, and Micro I/O System

The PLC I/O system provides the interface between the Series 90-30 PLC and user-supplied devices and equipment. Series 90-30 I/O is called Model 30 I/O. Model 30 I/O modules plug directly into slots in the CPU baseplate or into slots in any of the expansion baseplates for the Series 90-30 PLC Model 331 or higher. Model 331, 340, and 341 I/O systems support up to 49 Model 30 I/O modules (5 racks). Model 351 and 352 I/O systems support up to 79 Model 30 I/O modules (8 racks). The Series 90-30 PLC Model 311 or Model 313 5-slot baseplate supports up to 5 Model 30 I/O modules; the Model 323 10-slot baseplate supports up to 10 Model 30 I/O modules.

The I/O structure for the Series 90-30 PLC is shown in the following figure.

PLC I/O System

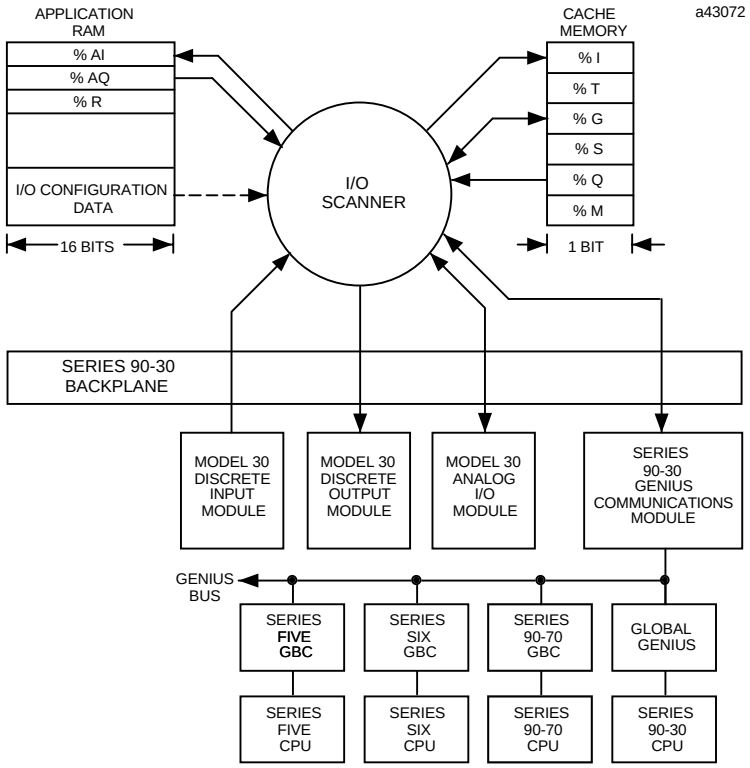


Figure 2-7. Series 90-30 I/O Structure

Note

The drawing shown above is specific to the 90-30 I/O structure. Intelligent and option modules are not part of the I/O scan; they use the System Communication Window. For information about the 90-20 I/O structure, refer to the *Series 90™-20 Programmable Controller User's Manual* (GFK-0551). For information about the Micro PLC I/O structure, refer to the *Series 90™ Micro PLC User's Manual* (GFK-1065).

Model 30 I/O Modules

Model 30 I/O modules are available as five types, discrete input, discrete output, analog input, analog output, and option modules. The following table lists the Model 30 I/O modules by catalog number, number of I/O points, and a brief description of each module.

Note

All of the I/O modules listed below may not be available at the time this manual is printed. For current availability, consult your local GE Fanuc PLC distributor or GE Fanuc sales representative. Refer to the *Series 90-30 I/O Module Specifications Manual*, GFK-0898, for the specifications and wiring information of each Model 30 I/O module.

Figure 2-8. Model 30 I/O Modules

Catalog Number	Points	Description	Pub Number
		Discrete Modules - Input	
IC693MDL230	8	120 VAC Isolated	GFK-0898
IC693MDL231	8	240 VAC Isolated	GFK-0898
IC693MDL240	16	120 VAC	GFK-0898
IC693MDL241	16	24 VAC/DC Positive/Negative Logic	GFK-0898
IC693MDL630	8	24 VDC Positive Logic	GFK-0898
IC693MDL632	8	125 VDC Positive/Negative Logic	GFK-0898
IC693MDL633	8	24 VDC Negative Logic	GFK-0898
IC693MDL634	8	24 VDC Positive/Negative Logic	GFK-0898
IC693MDL640	16	24 VDC Positive Logic	GFK-0898
IC693MDL641	16	24 VDC Negative Logic	GFK-0898
IC693MDL643	16	24 VDC Positive Logic, FAST	GFK-0898
IC693MDL644	16	24 VDC Negative Logic, FAST	GFK-0898
IC693MDL645	16	24 VDC Positive/Negative Logic	GFK-0898
IC693MDL646	16	24 VDC Positive/Negative Logic, FAST	GFK-0898
IC693MDL652	32	24 VDC Position/Negative Logic	GFK-0898
IC693MDL653	32	24 VDC Positive/Negative Logic, FAST	GFK-0898
IC693MDL654	32	5/12 VDC (TTL) Positive/Negative Logic	GFK-0898
IC693MDL655	32	24 VDC Positive/Negative Logic	GFK-0898
IC693ACC300	8/16	Input Simulator	GFK-0898

Table 2-7. Model 30 I/O Modules - Continued

Catalog Number	Points	Description	Pub Number
		Discrete Modules - Output	
IC693MDL310	12	120 VAC, 0.5A	GFK-0898
IC693MDL330	8	120/240 VAC, 2A	GFK-0898
IC693MDL340	16	120 VAC, 0.5A	GFK-0898
IC693MDL390	5	120/240 VAC Isolated, 2A	GFK-0898
IC693MDL730	8	12/24 VDC Positive Logic, 2A	GFK-0898
IC693MDL731	8	12/24 VDC Negative Logic, 2A	GFK-0898
IC693MDL732	8	12/24 VDC Positive Logic, 0.5A	GFK-0898
IC693MDL733	8	12/24 VDC Negative Logic, 0.5A	GFK-0898
IC693MDL734	6	125 VDC Positive/Negative Logic, 2A	GFK-0898
IC693MDL740	16	12/24 VDC Positive Logic, 0.5A	GFK-0898
IC693MDL741	16	12/24 VDC Negative Logic, 0.5A	GFK-0898
IC693MDL742	16	12/24 VDC Positive Logic, 1A	GFK-0898
IC693MDL750	32	12/24 VDC Negative Logic	GFK-0898
IC693MDL751	32	12/24 VDC Positive Logic, 0.3A	GFK-0898
IC693MDL752	32	5/24 VDC (TTL) Negative Logic, 0.5A	GFK-0898
IC693MDL753	32	12/24 VDC Positive/Negative Logic, 0.5A	GFK-0898
IC693MDL930	8	Relay, N.O., 4A Isolated	GFK-0898
IC693MDL931	8	Relay, BC, Isolated	GFK-0898
IC693MDL940	16	Relay, N.O., 2A	GFK-0898
		Input/Output Modules	
IC693MDR390	8/8	24 VDC Input, Relay Output	GFK-0898
IC693MAR590	8/8	120 VAC Input, Relay Output	GFK-0898
		Analog Modules	
IC693ALG220	4 ch	Analog Input, Voltage	GFK-0898
IC693ALG221	4 ch	Analog Input, Current	GFK-0898
IC693ALG222	16	Analog Input, Voltage	GFK-0898
IC693ALG223	16	Analog Input, Current	GFK-0898
IC693ALG390	2 ch	Analog Output, Voltage	GFK-0898
IC693ALG391	2 ch	Analog Output, Current	GFK-0898
IC693ALG392	8 ch	Analog Output, Current/Voltage	GFK-0898
IC693ALG442	4/2	Analog, Current/Voltage Combination Input/Output	GFK-0898

Table 2-7. Model 30 I/O Modules - Continued

Catalog Number	Description	Pub Number
	Option Modules	
IC693APU300	High Speed Counter	GFK-0293
IC693CMM311	Communications Coprocessor Module	GFK-0582
IC693PCM300	PCM, 160K Bytes (35KBytes User MegaBasic Program)	GFK-0255
IC693PCM301	PCM, 192K Bytes (47KBytes User MegaBasic Program)	GFK-0255
IC693PCM311	PCM, 640K Bytes (190KBytes User MegaBasic Program)	GFK-0255
IC693ADC311	Alphanumeric Display Coprocessor	GFK-0521
IC693BEM331	Genius Bus Controller	GFK-1034
IC693CMM301	Genius Communications Module	GFK-0412
IC693CMM302	Enhanced Genius Communications Module	GFK-0695
IC693BEM320	I/O Link Interface Module (slave)	GFK-0631
IC693BEM321	I/O Link Interface Module (master)	GFK-0823
IC693APU301	Power Mate APM Module, 1-Axis–Follower Mode	GFK-0781
IC693APU301	Power Mate APM Module, 1-Axis–Standard Mode	GFK-0840
IC693APU302	Power Mate APM Module, 2-Axis–Follower Mode	GFK-0781
IC693APU302	Power Mate APM Module, 2-Axis–Standard Mode	GFK-0840
IC693MCS001/2	Power Mate J Motion Control System (1 and 2 Axis)	GFK-1256
IC693APU305	I/O Processor Module	GFK-1028
IC693CMM321	Ethernet Communications Module	GFK-1084

I/O Data Formats

Discrete inputs and discrete outputs are stored as bits in bit cache (status table) memory. Analog input and analog output data are stored as words and are memory resident in a portion of application RAM memory allocated for that purpose.

Default Conditions for Model 30 Output Modules

At power-up, Model 30 discrete output modules default to outputs off. They will retain this default condition until the first output scan from the PLC. Analog output modules can be configured with a jumper located on the module's removable terminal block to either default to zero or retain their last state. Also, analog output modules may be powered from an external power source so that, even though the PLC has no power, the analog output module will continue to operate in its selected default state.

Diagnostic Data

Diagnostic bits are available in %S memory that will indicate the loss of an I/O module or a mismatch in I/O configuration. Diagnostic information is not available for individual I/O points. More information on fault handling can be in Chapter 3, "Fault Explanations and Correction."

Global Data

The Series 90-30 PLC supports very fast sharing of data between multiple CPUs using Genius global data. The Genius Bus Controller, IC693BEM331 in CPU, version 5 and later, and the Enhanced Genius Communications Module, IC693CMM302, can broadcast up to 128 bytes of data to other PLCs or computers. They can receive up to 128 bytes from each of the up to 30 other Genius controllers on the network. Data can be broadcast from or received into any memory type, not just %G global bits. The original Genius Communications Module, IC693CMM301, is limited to fixed %G addresses and can only exchange 32 bits per serial bus address from SBA 16 to 23. This module should not be used as the enhanced GCM has over 100 times the capability.

Global data can be shared between Series Five, Series Six, and Series 90 PLCs existing on the same Genius I/O bus.

Model 20 I/O Modules

The following I/O modules are available for the Series 90-20 PLC. Each module is listed by catalog number, number of I/O points, and a brief description. The I/O is integrated into a baseplate along with the power supply. For the specifications and wiring information of each module, refer to chapter 5 in the *Series 90-20 Programmable Controller User's Manua*l, GFK-0551.

Catalog Number	Description	I/O Points
IC692MAA541	I/O and Power Supply Base Module, 120 VAC In/120 VAC Out/120 VAC Power Supply	16 In/12 Out
IC692MDR541	I/O and Power Supply Base Module 24 VDC In/Relay Out/120 VAC Power Supply	16 In/12 Out
IC692MDR741	I/O and Power Supply Base Module 24V DC In/Relay Out/240 VAC Power Supply	16 In/12 Out
IC692CPU211	CPU Module, Model CPU 211	Not Applicable

Micro PLCs

The following Series 90 Micro PLCs are available. Each Micro is listed by catalog number, number of I/O points, and a brief description. The CPU, power supply, and I/O are all part of one unit. For the specifications and wiring information of each module, refer to the *Series 90 Micro Programmable Controller User's Manual*, GFK-1065.

Catalog Number	Description	I/O Points
IC693UDR001	CPU, Power Supply, and I/O (all one unit) Micro–14 pt. DC In/Relay Out, AC Power Supply	8 In/6 Out
IC693UDR002	CPU, Power Supply, and I/O (all one unit) Micro–14 pt. DC In/Relay Out, DC Power Supply	8 In/6 Out
IC693UAA003	CPU, Power Supply, and I/O (all one unit) Micro–14 pt. AC In/AC Out, AC Power Supply	8 In/6 Out
IC693UDD104	CPU, Power Supply, and I/O (all one unit) Micro–14 pt. DC In/DC Out, DC Power Supply	8 In/6 Out
IC693UDR005	CPU, Power Supply, and I/O (all one unit) Micro–28 pt. DC In/Relay Out, AC Power Supply	16 In/11 Relay Out/1 DC Out
IC693UAL006	CPU, Power Supply, and I/O (all one unit) Micro–23 pt. DC In/DC Out, AC Power Supply	1 DC Out/9 Relay Out/2 Analog In/1 Analog Out
IC693UAA007	CPU, Power Supply, and I/O (all one unit) Micro–28 pt. AC In/AC Out, AC Power Supply	16 In/12 Out
IC693UDR010	CPU, Power Supply, and I/O (all one unit) Micro–28 pt. DC In/DC Out, DC Power Supply	16 DC In/1 DC Out/11 Relay Out
IC693UEX011	14-point Expansion Unit–14 pt. DC In/Relay Out, AC Power Supply	8 In/6 Out

Chapter 3 Fault Explanation and Correction

This chapter is an aid to troubleshooting the Series 90-30, 90-20, and Micro PLC systems. It explains the fault descriptions, which appear in the PLC fault table, and the fault categories, which appear in the I/O fault table.

Each fault explanation in this chapter lists the fault description for the PLC fault table or the fault category for the I/O fault table. Find the fault description or fault category corresponding to the entry on the applicable fault table displayed on your programmer screen. Beneath it is a description of the cause of the fault along with instructions to correct the fault.

Chapter 3 contains the following sections:

Section	Title	Description	Page
1	Fault Handling	Describes the type of faults that may occur in the Series 90-30 and how they are displayed in the fault tables. Descriptions of the PLC and I/O fault table displays are also included.	3-2
2	PLC Fault Table Explanations	Provides a fault description of each PLC fault and instructions to correct the fault.	3-7
3	I/O Fault Table Explanations	Describes the Loss of I/O Module and Addition of I/O Module fault categories.	3-17

Section 1: Fault Handling

Note

This information on fault handling applies to systems programmed using Logicmaster 90-30/20/Micro software.

Faults occur in the Series 90-30 , 90-20, or Series 90 Micro PLC system when certain failures or conditions happen which affect the operation and performance of the system. These conditions, such as the loss of an I/O module or rack, may affect the ability of the PLC to control a machine or process. These conditions may also have beneficial effects, such as when a new module comes online and is now available for use. Or, these conditions may only act as an alert, such as a low battery signal which indicates that the battery protecting the memory needs to be changed.

Alarm Processor

The condition or failure itself is called a fault. When a fault is received and processed by the CPU, it is called an alarm. The software in the CPU which handles these conditions is called the Alarm Processor. The interface to the user for the Alarm Processor is through the programming software. Any detected fault is recorded in a fault table and displayed on either the PLC fault table screen or the I/O fault table screen, as applicable.

Classes of Faults

The Series 90-30, 90-20, and Micro PLCs detect several classes of faults. These include internal failures, external failures, and operational failures.

Fault Class	Examples
Internal Failures	Non-responding modules. Low battery condition. Memory checksum errors.
External I/O Failures	Loss of rack or module. Addition of rack or module.
Operational Failures	Communication failures. Configuration failures. Password access failures.

Note

For information specific to Micro PLC fault handling, refer to chapter 7 of the Series *90 Micro PLC User's Manual* (GFK-1065).

System Reaction to Faults

Typically, hardware failures require that either the system be shut down or the failure is tolerated. I/O failures may be tolerated by the PLC system, but they may be intolerable by the application or the process being controlled. Operational failures are normally tolerated. Series 90-30, 90-20, and Micro PLC faults have two attributes:

Attribute	Description
Fault Table Affected	I/O Fault Table PLC Fault Table
Fault Action	Fatal Diagnostic Informational

Fault Tables

Two fault tables are maintained in the PLC for logging faults, the I/O fault table for logging faults related to the I/O system and the PLC fault table for logging all other faults. The following table lists the fault groups, their fault actions, the fault tables affected, and the "name" for system discrete %S points that are affected.

Table 3-1. Fault Summary

Fault Group	Fault Action	Fault Table	Special Discrete Fault References			
Loss of or Missing I/O Module	Diagnostic	I/O	io_flt	any_flt	io_pres	los_iom
Loss of or Missing Option Module	Diagnostic	PLC	sy_flt	any_flt	sy_pres	los_sio
System Configuration Mismatch	Fatal	PLC	sy_flt	any_flt	sy_pres	cfg_mm
PLC CPU Hardware Failure	Fatal	PLC	sy_flt	any_flt	sy_pres	hrd_cpu
Program Checksum Failure	Fatal	PLC	sy_flt	any_flt	sy_pres	pb_sum
Low Battery	Diagnostic	PLC	sy_flt	any_flt	sy_pres	low_bat
PLC Fault Table Full	Diagnostic	—	sy_full			
I/O Fault Table Full	Diagnostic	—	io_full			
Application Fault	Diagnostic	PLC	sy_flt	any_flt	sy_pres	apl_flt
No User Program	Informational	PLC	sy_flt	any_flt	sy_pres	no_prog
Corrupted User RAM	Fatal	PLC	sy_flt	any_flt	sy_pres	bad_ram
Password Access Failure	Diagnostic	PLC	sy_flt	any_flt	sy_pres	bad_pwd
PLC Software Failure	Fatal	PLC	sy_flt	any_flt	sy_pres	sft_cpu
PLC Store Failure	Fatal	PLC	sy_flt	any_flt	sy_pres	stor_er
Constant Sweep Time Exceeded	Diagnostic	PLC	sy_flt	any_flt	sy_pres	ov_swp
Unknown PLC Fault	Fatal	PLC	sy_flt	any_flt	sy_pres	
Unknown I/O Fault	Fatal	I/O	io_flt	any_flt	io_pres	

Fault Action

Faults may be fatal, diagnostic or informational.

Fatal faults cause the fault to be recorded in the appropriate table, any diagnostic variables to be set, and the system to be halted. Diagnostic faults are recorded in the appropriate table, and any diagnostic variables are set. Informational faults are only recorded in the appropriate table.

Possible fault actions are listed in the following table.

Table 3-2. Fault Actions

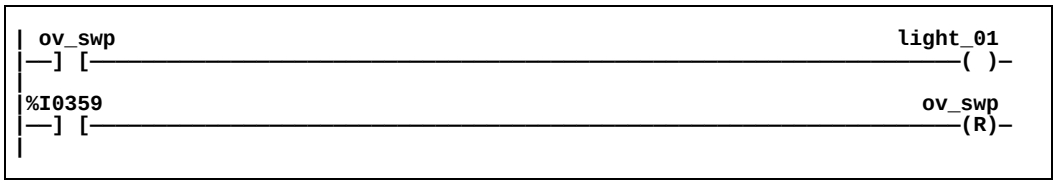
Fault Action	Response by CPU
Fatal	Log fault in fault table. Set fault references. Go to **STOP** mode.
Diagnostic	Log fault in fault table. Set fault references.
Informational	Log fault in fault table.

When a fault is detected, the CPU uses the fault action for that fault. Fault actions are not configurable in the Series 90-30 PLC, Series 90-20, or the Series 90 Micro PLC.

Fault References

Fault references in the Series 90-30 are of one type, fault summary references. Fault summary references are set to indicate what fault occurred. The fault reference remains on until the PLC is cleared or until cleared by the application program.

An example of a fault bit being set and then clearing the bit is shown in the following example. In this example, the coil light_01 is turned on when an oversweep condition occurs; the light and the OV_SWP contact remain on until the %I0359 contact is closed.



Fault Reference Definitions

The alarm processor maintains the states of the 128 system discrete bits in %S memory. These fault references can be used to indicate where a fault has occurred and what type of fault it is. Fault references are assigned to %S, %SA, %SB, and %SC memory, and they each have a nickname. These references are available for use in the application program as required. Refer to Chapter 2, "System Operation," for a list of the system status references.

Additional Fault Effects

Two faults described previously have additional effects associated with them. These are described in the following table.

Side Effect	Description
PLC CPU Software Failure	Whenever a PLC CPU software failure is logged, the Series 90-30 or 90-20 CPU immediately transitions into a special **ERROR SWEEP** mode. No activity is permitted in this mode. The only method of clearing this condition is to reset the PLC by cycling power.
PLC Sequence Store Failure	During a sequence store (a store of program blocks and other data preceded with the special Start-of-Sequence command and ending with the End-of-Sequence command), if communications with the programming device performing the store is interrupted or any other failure occurs which terminates the download, the PLC Sequence Store Failure fault is logged. As long as this fault is present in the system, the PLC will not transition to **RUN** mode.

PLC Fault Table Display

The PLC Fault Table screen displays PLC faults such as password violations, PLC/configuration mismatches, parity errors, and communications errors.

The programming software may be in any operating mode. If the programming software is in **OFFLINE** mode, no faults are displayed. In **ONLINE** or **MONITOR** mode, PLC fault data is displayed. In **ONLINE** mode, faults can be cleared (this may be password protected).

Once cleared, faults which are still present are not logged again in the table (except for the "Low Battery" fault).

I/O Fault Table Display

The I/O Fault Table screen displays I/O faults such as circuit faults, address conflicts, forced circuits, and I/O bus faults.

The programming software may be in any operating mode. If the programming software is in **OFFLINE** mode, no faults are displayed. In **ONLINE** or **MONITOR** mode, I/O fault data is displayed. In **ONLINE** mode, faults can be cleared (this feature may be password protected).

Once cleared, faults which are still present are not logged again in the table.

Accessing Additional Fault Information

The fault tables contain basic information regarding the fault. Additional information pertaining to each fault can be displayed through the programming software. In addition, the programming software can provide a hexadecimal dump of the fault.

The last entry, Correction, for each fault explanation in this chapter lists the action(s) to be taken to correct the fault. Note that the corrective action for some of the faults includes the statement:

Display the PLC Fault Table on the Programmer. Contact GE Fanuc Field Service, giving them all the information contained in the fault entry.

This second statement means that you must tell Field Service both the information readable directly from the fault table **and** the hexadecimal information. Field Service personnel will then give you further instructions for the appropriate action to be taken.

Section 2: PLC Fault Table Explanations

Each fault explanation contains a fault description and instructions to correct the fault. Many fault descriptions have multiple causes. In these cases, the error code, displayed with the additional fault information, is used to distinguish different fault conditions sharing the same fault description. The error code is the first two hexadecimal digits in the fifth group of numbers, as shown in the following example.

```
01   000000   01030100   0902   0200   000000000000
                                 |
                                 |_____  Error Code  (first two hex
                                              digits in fifth group)
```

Some faults can occur because random access memory on the PLC CPU board has failed. These same faults may also occur because the system has been powered off and the battery voltage is (or was) too low to maintain memory. To avoid excessive duplication of instructions when corrupted memory may be a cause of the error, the correction simply states:

`Perform the corrections for Corrupted Memory.`

This means:

1. If the system has been powered off, replace the battery. Battery voltage may be insufficient to maintain memory contents.
2. Replace the PLC CPU board. The integrated circuits on the PLC CPU board may be failing.

The following table enables you to quickly find a particular PLC fault explanation in this section. Each entry is listed as it appears on the programmer screen.

Fault Description	Page
Loss of, or Missing, Option Module	3-8
Reset of, Addition of, or Extra, Option Module	3-9
System Configuration Mismatch	3-10
Option Module Software Failure	3-11
Program Block Checksum Failure	3-11
Low Battery Signal	3-11
Constant Sweep Time Exceeded	3-12
Application Fault	3-12
No User Program Present	3-13
Corrupted User Program on Power-Up	3-13
Password Access Failure	3-13
PLC CPU System Software Failure	3-14
Communications Failure During Store	3-16

Fault Actions

Fatal faults cause the PLC to enter a form of **STOP** mode at the end of the sweep in which the error occurred. **Diagnostic** faults are logged and corresponding fault contacts are set. **Informational** faults are simply logged in the PLC fault table.

Loss of, or Missing, Option Module

The Fault Group **Loss of, or Missing Option Module** occurs when a PCM, CMM, or ADC fails to respond. The failure may occur at power-up if the module is missing or during operation if the module fails to respond. The fault action for this group is **Diagnostic**.

Error Code:	1, 42
Name:	Option Module Soft Reset Failed
Description:	PLC CPU unable to re-establish communications with option module after soft reset.
Correction:	(1) Try soft reset a second time. (2) Replace the option module. (3) Power off the system. Verify that the PCM is seated properly in the rack and that all cables are properly connected and seated. (4) Replace the cables.
Error Code:	79
Name:	Loss of Daughterboard
Description:	The daughterboard has been lost (i.e., not seen by the CPU) and will not function.
Correction:	Make sure the CPU has a daughterboard physically present.
Error Code:	All Others
Name:	Module Failure During Configuration
Description:	The PLC operating software generates this error when a module fails during power-up or configuration store.
Correction:	Power off the system. Replace the module located in that rack and slot.

Reset of, Addition of, or Extra, Option Module

The Fault Group **Reset of, Addition of, or Extra Option Module** occurs when an option module (PCM, ADC, etc.) comes online, is reset, or a module is found in the rack, but none is specified in the configuration. The fault action for this group is **Diagnostic**. Three bytes of fault specific data provide additional information regarding the fault.

Correction:	(1) Update the configuration file to include the module. (2) Remove the module from the system.

This Fault Group also includes the following faults specific to systems having a daughterboard:

Error Code:	4
Name:	Addition of Daughterboard
Description:	There is a daughterboard present but not configured.
Correction:	Make sure the configuration stored to the CPU contains the correct daughterboard.
Error Code:	5
Name:	Daughterboard Reset
Description:	The daughterboard has been reset either due to the occurance of a push button reset by the user or an internal error condition.
Correction:	None

System Configuration Mismatch

The Fault Group **Configuration Mismatch** occurs when the module occupying a slot is different from that specified in the configuration file. The fault action is **Fatal**.

Error Code:	1
Name:	System Configuration Mismatch
Description:	The PLC operating software (system configurer) generates this fault when the module occupying a slot is not of the same type that the configuration file indicates should be in that slot, or when the configured rack type does not match the actual rack present.
Correction:	Identify the mismatch and reconfigure the module or rack.
Error Code:	6
Name:	System Configuration Mismatch
Description:	This is the same as error code 1 in that this fault occurs when the module occupying a slot is not of the same type that the configuration file indicates should be in that slot, or when the configured rack type does not match the actual rack present.
Correction:	Identify the mismatch and reconfigure the module or rack.
Error Code:	18
Name:	Unsupported Hardware
Description:	A PCM or PCM-type module is present in a 311, 313, or 323, or in an extension rack.
Correction:	Physically correct the situation by removing the PCM or PCM-type module or install a CPU that does support the PCM.
Error Code:	26
Name:	Module busy–config not yet accept by module
Description:	The module cannot accept new configuration at this time because it is busy with a different process.
Correction:	Allow the module to complete the current operation and re-store the configuration.
Error Code:	51
Name:	END Function Executed from SFC Action
Description:	The placement of an END function in SFC logic or in logic called by SFC will produce this fault.
Correction:	Remove the END function from the SFC logic or logic being called by the SFC logic.
Error Code:	58
Name:	Daughterboard Mismatch
Description:	The daughterboard physically present does not match the daughterboard in the configuration.
Correction:	Make sure the configuration stored to the CPU contains the correct daughterboard.

Option Module Software Failure

The Fault Group **Option Module Software Failure** occurs when a non-recoverable software failure occurs on a PCM or ADC module. The fault action for this group is **Fatal**.

Error Code:	All
Name:	COMMREQ Frequency Too High
Description:	COMMREQs are being sent to a module faster than it can process them.
Correction:	Change the PLC program to send COMMREQs to the affected module at a slower rate.

Program Block Checksum Failure

The Fault Group **Program Block Checksum Failure** occurs when the PLC CPU detects error conditions in program blocks received by the PLC. It also occurs when the PLC CPU detects checksum errors during power-up verification of memory or during **RUN** mode background checking. The fault action for this group is **Fatal**.

Error Code:	All
Name:	Program Block Checksum Failure
Description:	The PLC Operating Software generates this error when a program block is corrupted.
Correction:	(1) Clear PLC memory and retry the store. (2) Display the PLC fault table on the programmer. Contact GE Fanuc PLC Field Service, giving them all the information contained in the fault entry.

Low Battery Signal

The Fault Group **Low Battery Signal** occurs when the PLC CPU detects a low battery on the PLC power supply or a module, such as the PCM, reports a low battery condition. The fault action for this group is **Diagnostic**.

Error Code:	0
Name:	Failed Battery Signal
Description:	The CPU module (or other module having a battery) battery is dead.
Correction:	Replace the battery. Do not remove power from the rack.
Error Code:	1
Name:	Low Battery Signal
Description:	A battery on the CPU, or other module has a low signal.
Correction:	Replace the battery. Do not remove power from the rack.

Constant Sweep Time Exceeded

The Fault Group **Constant Sweep Time Exceeded** occurs when the PLC CPU operates in **CONSTANT SWEEP** mode, and it detects that the sweep has exceeded the constant sweep timer. The fault extra data contains the actual time of the sweep in the first two bytes and the name of the program in the next eight bytes. The fault action for this group is **Diagnostic**.

Correction:	(1)	Increase constant sweep time.
	(2)	Remove logic from application program.

Application Fault

The Fault Group **Application Fault** occurs when the PLC CPU detects a fault in the user program. The fault action for this group is **Diagnostic**, except when the error is a Subroutine Call Stack Exceeded, in which case it is **Fatal**.

Error Code:	7
Name:	Subroutine Call Stack Exceeded
Description:	Subroutine calls are limited to a depth of 8. A subroutine can call another subroutine which, in turn, can call another subroutine until 8 call levels are attained.
Correction:	Modify program so that subroutine call depth does not exceed 8.
Error Code:	1B
Name:	Comm Req Not Processed Due To PLC Memory Limitations
Description:	No-wait communication requests can be placed in the queue faster than they can be processed (e.g., one per sweep). In a situation like this, when the communication requests build up to the point that the PLC has less than a minimum amount of memory available, the communication request will be faulted and not processed
Correction:	Issue fewer communication requests or otherwise reduce the amount of mail being exchanged within the system.
Error Code:	5A
Name:	User Shut Down Requested
Description:	The PLC operating software (function blocks) generates this informational alarm when Service Request #13 (User Shut Down) executes in the application program.
Correction:	None required. Information-only alarm.

No User Program Present

The Fault Group **No User Program Present** occurs when the PLC CPU is instructed to transition from **STOP** to **RUN** mode or a store to the PLC and no user program exists in the PLC. The PLC CPU detects the absence of a user program on power-up. The fault action for this group is **Informational**.

Correction:	Download an application program before attempting to go to **RUN** mode.

Corrupted User Program on Power-Up

The Fault Group **Corrupted User Program on Power-Up** occurs when the PLC CPU detects corrupted user RAM. The PLC CPU will remain in **STOP** mode until a valid user program and configuration file are downloaded. The fault action for this group is **Fatal**.

Error Code:	1
Name:	Corrupted User RAM on Power-Up
Description:	The PLC operating software (operating software) generates this error when it detects corrupted user RAM on power-up.
Correction:	(1) Reload the configuration file, user program, and references (if any). (2) Replace the battery on the PLC CPU. (3) Replace the expansion memory board on the PLC CPU. (4) Replace the PLC CPU.
Error Code:	2
Name:	Illegal Boolean OpCode Detected
Description:	The PLC operating software (operating software) generates this error when it detects a bad instruction in the user program.
Correction:	(1) Restore the user program and references (if any). (2) Replace the expansion memory board on the PLC CPU. (3) Replace the PLC CPU.

Password Access Failure

The Fault Group **Password Access Failure** occurs when the PLC CPU receives a request to change to a new privilege level and the password included with the request is not valid for that level. The fault action for this group is **Informational**.

Correction:	Retry the request with the correct password.

PLC CPU System Software Failure

Faults in the Fault Group **PLC CPU System Software Failure** are generated by the operating software of the Series 90-30, 90-20 or Micro PLC CPU. They occur at many different points of system operation. When a **Fatal** fault occurs, the PLC CPU **immediately** transitions into a special **ERROR SWEEP** mode. No activity is permitted when the PLC is in this mode. The only way to clear this condition is to cycle power on the PLC. The fault action for this group is **Fatal**.

Error Code:	1 through B
Name:	User Memory Could Not Be Allocated
Description:	The PLC operating software (memory manager) generates these errors when software requests the memory manager to allocate or deallocate a block or blocks of memory from user RAM that are not legal. These errors should *not* occur in a production system.
Correction:	Display the PLC fault table on the programmer. Contact GE Fanuc PLC Field Service, giving them all the information contained in the fault entry.
Error Code:	D
Name:	System Memory Unavailable
Description:	The PLC operating software (I/O Scanner) generates this error when its request for a block of system memory is denied by the memory manager because no memory is available from the system memory heap. It is *Informational* if the error occurs during the execution of a DO I/O function block. It is *Fatal* if it occurs during power-up initialization or autoconfiguration.
Correction:	Display the PLC fault table on the programmer. Contact GE Fanuc PLC Field Service, giving them all the information contained in the fault entry.
Error Code:	E
Name:	System Memory Could Not Be Freed
Description:	The PLC operating software (I/O Scanner) generates this error when it requests the memory manager to deallocate a block of system memory and the deallocation fails. This error can only occur during the execution of a DO I/O function block.
Correction:	(1) Display the PLC fault table on the programmer. Contact GE Fanuc PLC Field Service, giving them all the information contained in the fault entry. (2) Perform the corrections for corrupted memory.
Error Code:	10
Name:	Invalid Scan Request of the I/O Scanner
Description:	The PLC operating software (I/O Scanner) generates this error when the operating system or DO I/O function block scan requests neither a full nor a partial scan of the I/O. This should *not* occur in a production system.
Correction:	Display the PLC fault table on the programmer. Contact GE Fanuc PLC Field Service, giving them all the information contained in the fault entry.
Error Code:	13
Name:	PLC Operating Software Error
Description:	The PLC operating software generates this error when certain PLC operating software problems occur. This error should *not* occur in a production system.
Correction:	(1) Display the PLC fault table on the programmer. Contact GE Fanuc PLC Field Service, giving them all the information contained in the fault entry. (2) Perform the corrections for corrupted memory.

Error Code:	14, 27
Name:	Corrupted PLC Program Memory
Description:	The PLC operating software generates these errors when certain PLC operating software problems occur. These should *not* occur in a production system.
Correction:	(1) Display the PLC fault table on the programmer. Contact GE Fanuc PLC Field Service, giving them all the information contained in the fault entry. (2) Perform the corrections for corrupted memory.
Error Code:	27 through 4E
Name:	PLC Operating Software Error
Description:	The PLC operating software generates these errors when certain PLC operating software problems occur. These errors should *not* occur in a production system.
Correction:	Display the PLC fault table on the programmer. Contact GE Fanuc PLC Field Service, giving them all the information contained in the fault entry.
Error Code:	4F
Name:	Communications Failed
Description:	The PLC operating software (service request processor) generates this error when it attempts to comply with a request that requires backplane communications and receives a rejected response.
Correction:	(1) Check the bus for abnormal activity. (2) Replace the intelligent option module to which the request was directed.
Error Code:	50, 51, 53
Name:	System Memory Errors
Description:	The PLC operating software generates these errors when its request for a block of system memory is denied by the memory manager because no memory is available or contains errors.
Correction:	(1) Display the PLC fault table on the programmer. Contact GE Fanuc PLC Field Service, giving them all the information contained in the fault entry. (2) Perform the corrections for corrupted memory.
Error Code:	52
Name:	Backplane Communications Failed
Description:	The PLC operating software (service request processor) generates this error when it attempts to comply with a request that requires backplane communications and receives a rejected mail response.
Correction:	(1) Check the bus for abnormal activity. (2) Replace the intelligent option module to which the request was directed. (3) Check parallel programmer cable for proper attachment.
Error Code:	All Others
Name:	PLC CPU Internal System Error
Description:	An internal system error has occurred that should **not** occur in a production system.
Correction:	Display the PLC fault table on the programmer. Contact GE Fanuc PLC Field Service, giving them all the information contained in the fault entry.

Communications Failure During Store

The Fault Group **Communications Failure During Store** occurs during the store of program blocks and other data to the PLC. The stream of commands and data for storing program blocks and data starts with a special start-of-sequence command and terminates with an end-of-sequence command. If communications with the programming device performing the store is interrupted or any other failure occurs which terminates the load, this fault is logged. As long as this fault is present in the system, the controller will not transition to **RUN** mode.

This fault is *not* automatically cleared on power-up; the user must specifically order the condition to be cleared. The fault action for this group is **Fatal**.

Correction:	Clear the fault and retry the download of the program or configuration file.

Section 3: I/O Fault Table Explanations

The I/O fault table reports data about faults in three classifications:

- Fault category.
- Fault type.
- Fault description.

The faults described on the following page have a fault category, but do not have a fault type or fault group.

Each fault explanation contains a fault description and instructions to correct the fault. Many fault descriptions have multiple causes. In these cases, the error code, displayed with the additional fault information obtained by pressing CTRL-F, is used to distinguish different fault conditions sharing the same fault description. (For more information about using CTRL-F, refer to Appendix B, "Interpreting Fault Tables," in this manual.) The Fault Category is the first two hexadecimal digits in the fifth group of numbers, as shown in the following example.

```
02  1F0100  00030101FF7F  0302  0200   84000000000003
                                 |
                                 |_____  Fault Category (first two hex
                                              digits in fifth group)
```

The following table enables you to quickly find a particular I/O fault explanation in this section. Each entry is listed as it appears on the programmer screen.

Loss of I/O Module

The Fault Category **Loss of I/O Module** applies to Model 30 discrete and analog I/O modules. There are no fault types or fault descriptions associated with this category. The fault action is **Diagnostic**.

Description:	The PLC operating software generates this error when it detects that a Model 30 I/O module is no longer responding to commands from the PLC CPU, or when the configuration file indicates an I/O module is to occupy a slot and no module exists in the slot.
Correction:	(1) Replace the module. (2) Correct the configuration file. (3) Display the PLC fault table on the programmer. Contact GE Fanuc PLC Field Service, giving them all the information contained in the fault entry.

Addition of I/O Module

The Fault Category **Addition of I/O Module** applies to Model 30 discrete and analog I/O modules. There are no fault types or fault descriptions associated with this category. The fault action is **Diagnostic**.

Description:	The PLC operating software generates this error when an I/O module which had been faulted returns to operation.
Correction:	(1) No action necessary if the module was removed or replaced, or the remote rack was power cycled. (2) Update the configuration file or remove the module.
Description:	The PLC operating software generates this error when it detects a Model 30 I/O module in a slot which the configuration file indicates should be empty.
Correction:	(1) Remove the module. (It may be in the wrong slot.) (2) Update and restore the configuration file to include the extra module.

Chapter 4 Series 90-30/20/Micro Instructions Set

Programming consists of creating an application program for a PLC. Because the Series 90-30, 90-20, and Series 90 Micro PLCs have a common instruction set, all three can be programmed using this software. This chapter describes the programming instructions that may be used to create ladder logic programs for the Series 90-30 and Series 90-20 programmable controllers.

If Logicmaster 90-30/20/Micro programming software is not yet installed, please refer to the *Programming Software User's Manual*, GFK-0466, for instructions. The user's manual explains how to create, transfer, edit, and print programs.

Configuration is the process of assigning logical addresses, as well as other characteristics, to the hardware modules in the system. It may be done either before or after programming, using the configuration software or Hand-Held Programmer; however, it is recommended that configuration be done first. If that has not been done, you should refer to the *Programming Software User's Manual*, GFK-0466, to decide whether it is best to begin programming at this time.

This chapter contains the following sections:

Section	Title	Description	Page
1	Relay Functions	Describes contacts, coils, and links.	4-2
2	Timers and Counters	Describes on-delay and stopwatch-type timers, up counters, and down counters.	4-9
3	Math Functions	Describes addition, subtraction, multiplication, division, modulo division, square root, trigonometric functions, logarithmic/exponential functions, and radian conversion. *Note that trigonometric functions, logarithmic/exponential functions, and radian conversion functions are only available with the 350 and 360 series of CPUs.*	4-26
4	Relational Functions	Describes how to compare two numbers for equality, non-equality, greater than, greater than or equal to, less than, and less than or equal to.	4-41
5	Bit Operation Functions	Describes how to perform comparison and move operations on bit strings.	4-47
6	Data Move Functions	Describes basic data move capabilities.	4-69
7	Table Functions	Describes how to use table functions to enter values into and copy values out of a table.	4-86
8	Conversion Functions	Describes how to convert a data item from one number type to another.	4-94
9	Control Functions	Describes how to limit program execution and alter the way the CPU executes the application program by using the control functions.	4-107

Section 1: Relay Functions

This section explains the use of contacts, coils, and links in ladder logic rungs.

Function	Page
Coils and negated coils.	4-3
Normally open and normal closed contacts.	4-2
Retentive and negated retentive coils.	4-4
Positive and negative transition coils.	4-5
SET and RESET coils.	4-6
Retentive SET and RESET coils.	4-7
Horizontal and vertical links.	4-7
Continuation coils and contacts.	4-8

Using Contacts

A contact is used to monitor the state of a reference. Whether the contact passes power flow depends on the state or status of the reference being monitored and on the contact type. A reference is ON if its state is 1; it is OFF if its state is 0.

Table 4-1. Types of Contacts

Type of Contact	Display	Contact Passes Power to Right:
Normally Open	—| |—	When reference is ON.
Normally Closed	—|/|—	When reference is OFF.
Continuation Contact	<+>———	If the preceding continuation coil is set ON.

Using Coils

Coils are used to control discrete references. Conditional logic must be used to control the flow of power to a coil. Coils cause action directly; they do not pass power flow to the right. If additional logic in the program should be executed as a result of the coil condition, an internal reference should be used for that coil or a continuation coil/contact combination may be used.

Coils are always located at the rightmost position of a line of logic. A rung may contain up to eight coils.

The type of coil used will depend on the type of program action desired. The states of retentive coils are saved when power is cycled or when the PLC goes from **STOP** to **RUN** mode. The states of non-retentive coils are set to zero when power is cycled or the PLC goes from **STOP** to **RUN** mode.

Table 4-2. Types of Coils

Type of Coil	Display	Power to Coil	Result
Normally Open	—()—	ON OFF	Set reference ON. Set reference OFF.
Negated	—(/)—	ON OFF	Set reference OFF. Set reference ON.
Retentive	—(M)—	ON OFF	Set reference ON, retentive. Set reference OFF, retentive.
Negated Retentive	—(/M)—	ON OFF	Set reference OFF, retentive. Set reference ON, retentive.
Positive Transition	—(↑)—	OFF→ON	If reference is OFF,set it ON for one sweep.
Negative Transition	—(↓)—	ON←OFF	If reference is OFF, set it ON for one sweep.
SET	—(S)—	ON OFF	Set reference ON until reset OFF by —(R)—. Do not change the coil state.
RESET	—(R)—	ON OFF	Set reference OFF until set ON by —(S)—. Do not change the coil state.
Retentive SET	—(SM)—	ON OFF	Set reference ON until reset OFF by —(RM)—, retentive. Do not change the coil state.
Retentive RESET	—(RM)—	ON OFF	Set reference OFF until set ON by —(SM)—, retentive. Do not change the coil state.
Continuation Coil	——<+>	ON OFF	Set next continuation contact ON. Set next continuation contact OFF.

Normally Open Contact —| |—

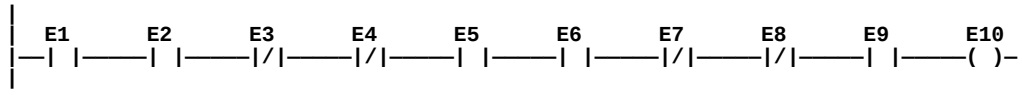
A normally open contact acts as a switch that passes power flow if the associated reference is ON (1).

Normally Closed Contact —|/|—

A normally closed contact acts as a switch that passes power flow if the associated reference is OFF (0).

Example:

The following example shows a rung with 10 elements having nicknames from E1 to E10. Coil E10 is ON when reference E1, E2, E5, E6, and E9 are ON and references E3, E4, E7, and E8 are OFF.

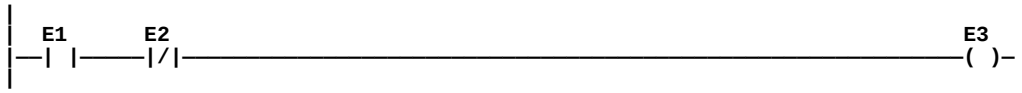


Coil —( )—

A coil sets a discrete reference ON while it receives power flow. It is non-retentive; therefore, it cannot be used with system status references (%SA, %SB, %SC, or %G).

Example:

In the following example, coil E3 is ON when reference E1 is ON and reference E2 is OFF.

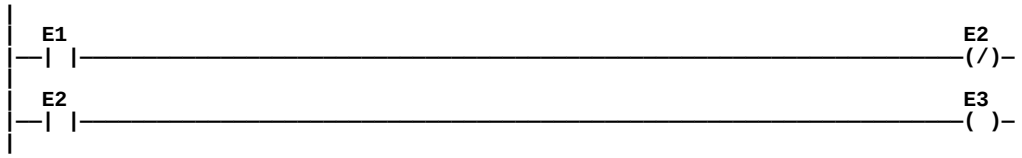


Negated Coil —(/)—

A negated coil sets a discrete reference ON when it does not receive power flow. It is not retentive; therefore, it cannot be used with system status references (%SA, %SB, %SC, or %G).

Example:

In the following example, coil E3 is ON when reference E1 is OFF.



Retentive Coil —(M)—

Like a normally open coil, the retentive coil sets a discrete reference ON while it receives power flow. The state of the retentive coil is retained across power failure. Therefore, it cannot be used with references from strictly non-retentive memory (%T).

Negated Retentive Coil —(/M)—

The negated retentive coil sets a discrete reference ON when it does not receive power flow. The state of the negated retentive coil is retained across power failure. Therefore, it cannot be used with references from strictly non-retentive memory (%T).

Positive Transition Coil —(↑)—

If the reference associated with a positive transition coil is OFF, when the coil receives power flow it is set to ON until the next time the coil is executed. (If the rung containing the coil is skipped on subsequent sweeps, it will remain ON.) This coil can be used as a one-shot.

Each reference should only be used as a transition coil once in the application program, so as to preserve the one-shot nature of the coil.

Transitional coils can be used with references from either retentive or non-retentive memory (%Q, %M, %T, %G, %SA, %SB, or %SC).

Negative Transition Coil —(↓)—

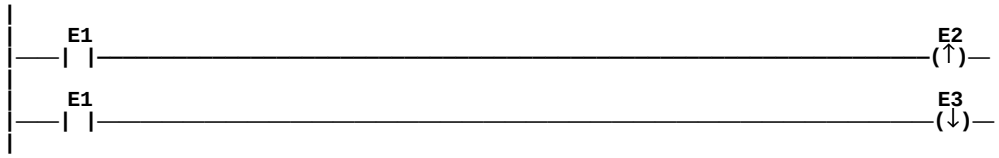
If the reference associated with this coil is OFF, when the coil ***stops*** receiving power flow, the reference is set to ON until the next time the coil is executed.

Each reference should only be used as a transition coil once in the application program, so as to preserve the one-shot nature of the coil.

Transitional coils can be used with references from either retentive or non-retentive memory (%Q, %M, %T, %G, %SA, %SB, or %SC).

Example:

In the following example, when reference E1 goes from OFF to ON, coils E2 and E3 receive power flow, turning E2 ON for one logic sweep. When E1 goes from ON to OFF, power flow is removed from E2 and E3, turning coil E3 ON for one sweep.



SET Coil —(S)—

SET and RESET are non-retentive coils that can be used to keep (“latch”) the state of a reference (e.g., E1) either ON or OFF. When a SET coil receives power flow, its reference stays ON (whether or not the coil itself receives power flow) until the reference is reset by another coil.

SET coils write an undefined result to the transition bit for the given reference. (Refer to the information on “Transitions and Overrides” in chapter 2, “System Operation.”)

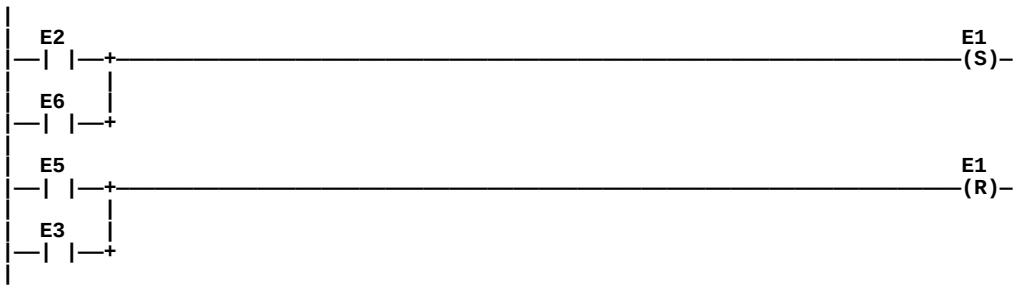
RESET Coil —(R)—

The RESET coil sets a discrete reference OFF if the coil receives power flow. The reference remains OFF until the reference is reset by another coil. The last-solved SET coil or RESET coil of a pair takes precedence.

RESET coils write an undefined result to the transition bit for the given reference. (Refer to the information on “Transitions and Overrides” in chapter 2, “System Operation.”)

Example:

In the following example, the coil represented by E1 is turned ON whenever reference E2 or E6 is ON. The coil represented by E1 is turned OFF whenever reference E5 or E3 is ON.



Note

When the level of coil checking is SINGLE, you can use a specific %M or %Q reference with only one Coil, but you can use it with one SET Coil and one RESET Coil simultaneously. When the level of coil checking is WARN MULTIPLE or MULTIPLE, then each reference can be used with multiple Coils, SET Coils, and RESET Coils. With multiple usage, a reference could be turned ON by either a SET Coil or a normal Coil and could be turned OFF by a RESET Coil or by a normal Coil.

Retentive SET Coil —(SM)—

Retentive SET and RESET coils are similar to SET and RESET coils, but they are retained across power failure or when the PLC transitions from **STOP** to **RUN** mode. A retentive SET coil sets a discrete reference ON if the coil receives power flow. The reference remains ON until reset by a retentive RESET coil.

Retentive SET coils write an undefined result to the transition bit for the given reference. (Refer to the information on “Transitions and Overrides” in chapter 2, “System Operation.”)

Retentive RESET Coil —(RM)—

This coil sets a discrete reference OFF if it receives power flow. The reference remains OFF until set by a retentive SET coil. The state of this coil is retained across power failure or when the PLC transitions from **STOP** to **RUN** mode.

Retentive RESET coils write an undefined result to the transition bit for the given reference. (Refer to the information on “Transitions and Overrides” in chapter 2, “System Operation.”)

Links

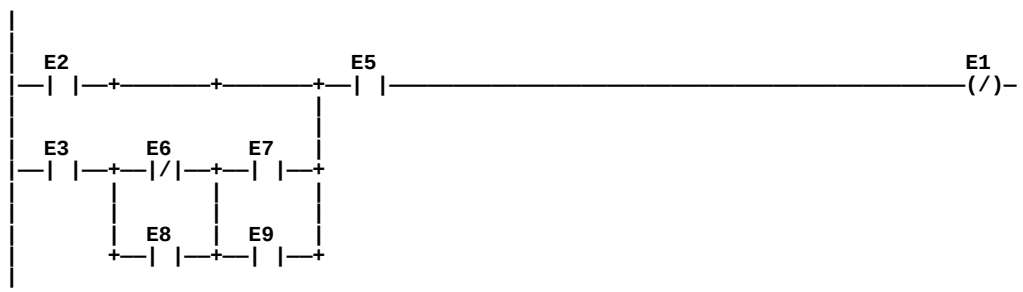
Horizontal and vertical links are used to connect elements of a line of ladder logic between functions. Their purpose is to complete the flow of logic (“power”) from left to right in a line of logic.

Note

You can not use a horizontal link to tie a function or coil to the left power rail. You can, however, use %S7, the AWL_ON (always on) system bit with a normally open contact tied to the power rail to call a function every sweep.

Example:

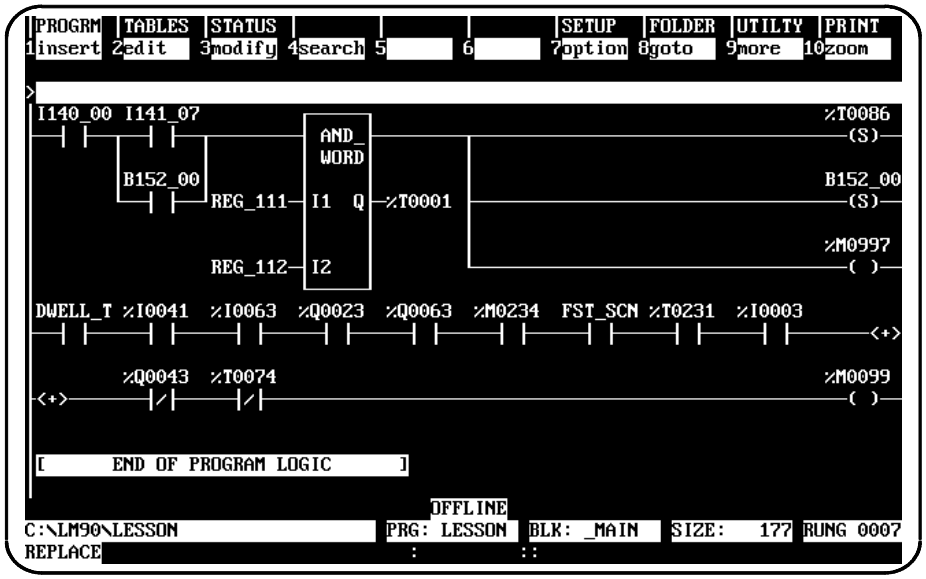
In the following example, two horizontal links are used to connect contacts E2 and E5. A vertical link is used to connect contacts E3, E6, E7, E8, and E9 to E2.



Continuation Coils (———<+>) and Contacts (<+>———)

Continuation coils (———<+>) and continuation contacts (<+>———) are used to continue relay ladder rung logic beyond the limit of ten columns. The state of the last executed continuation coil is the flow state that will be used on the next executed continuation contact. There needs to be a continuation coil before the logic executes a continuation contact. The state of the continuation contact is cleared when the PLC transitions from **Stop** to **Run**, and there will be no flow unless the transition coil has been set since going to **Run** mode.

There can be only one continuation coil and contact per rung; the continuation contact must be in column 1, and the continuation coil must be in column 10. An example continuation coil and contact are shown below:



Section 2: Timers and Counters

This section explains how to use on-delay and stopwatch-type timers, up counters, and down counters. The data associated with these functions is retentive through power cycles.

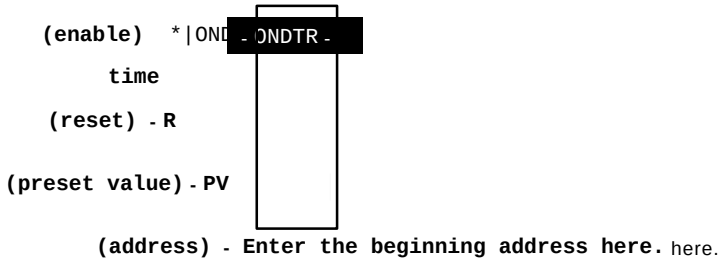
Abbreviation	Function	Page
ONDTR	Retentive On-Delay Timer	4-11
TMR	Simple On-Delay Timer	4-14
OFDT	Off-Delay Timer	4-17
UPCTR	Up Counter	4-20
DNCTR	Down Counter	4-22

Function Block Data Required for Timers and Counters

Each timer or counter uses three words (registers) of %R memory to store the following information:

current value (CV)	word 1
preset value (PV)	word 2
control word	word 3

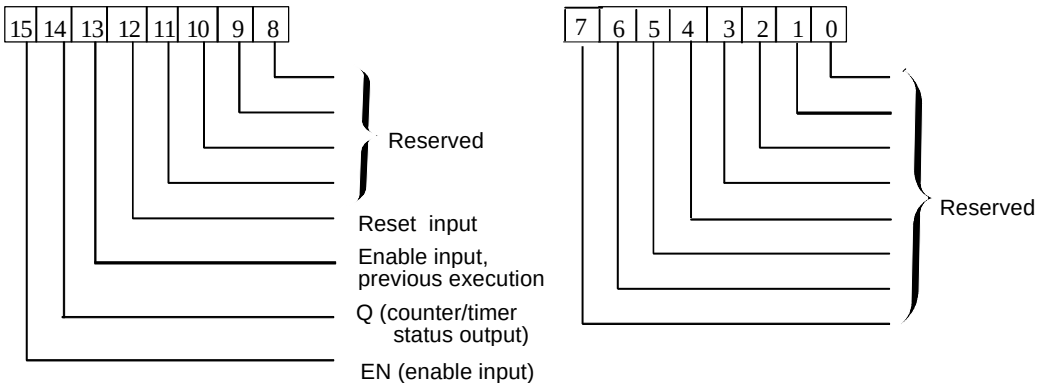
When you enter a timer or counter, you must enter a beginning address for these three words (registers) directly below the graphic representing the function. For example:



Note

Do not use consecutive registers for the 3 word timer/counter blocks. Logicmaster does ***not*** check or warn you if register blocks overlap. Timers and counters will not work if you place the current value of a block on top of the preset for the previous block.

The control word stores the state of the boolean inputs and outputs of its associated function block, as shown in the following format:



Bits 0 through 11 are used for timer accuracy; bits 0 through 11 are not used for counters.

Note

Use care if you use the same address for PV as the second word in the block of three words. If PV is not a constant, the PV is normally set to a different location than the second word. Some applications choose to use the second word address for the PV, such as using %R0102 when the bottom data block starts at %R0101. This allows an application to change the PV while the timer or counter is running. Applications can read the first CV or third Control words, but the application cannot write to these values, or the function will not work.

Special Note on Certain Bit Operations

When using the Bit Test, Bit Set, Bit Clear or Bit Position function, the bits are numbered 1 through 16, ***NOT*** 0 through 15 as shown above.

ONDTR

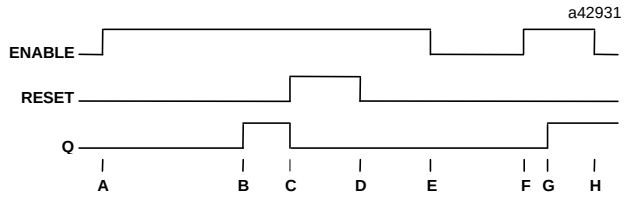
A retentive on-delay timer (ONDTR) increments while it receives power flow and holds its value when power flow stops. Time may be counted in tenths of a second (the default selection), hundredths of a second, or thousandths of a second. The range is 0 to +32,767 time units. The state of this timer is retentive on power failure; no automatic initialization occurs at power-up.

When the ONDTR first receives power flow, it starts accumulating time (current value). When this timer is encountered in the ladder logic, its current value is updated.

Note

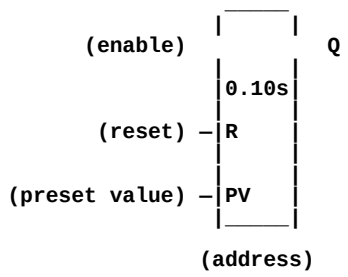
If multiple occurrences of the same timer with the same reference address are enabled during a CPU sweep, the current values of the timers will be the same.

When the current value equals or exceeds the preset value PV, output Q is energized. As long as the timer continues to receive power flow, it continues accumulating until the maximum value is reached. Once the maximum value is reached, it is retained and output Q remains energized regardless of the state of the enable input.



A = ENABLE goes high; timer starts accumulating.
B = CV reaches PV; Q goes high.
C = RESET goes high; Q goes low, accumulated time is reset.
D = RESET goes low; timer then starts accumulating again.
E = ENABLE goes low; timer stops accumulating. Accumulated time stays the same.
F = ENABLE goes high again; timer continues accumulating time.
G = CV becomes equal to PV; Q goes high. Timer continues to accumulate time until ENABLE goes low, RESET goes high, or CV becomes equal to the maximum time.
H = ENABLE goes low; timer stops accumulating time.

When power flow to the timer stops, the current value stops incrementing and is retained. Output Q, if energized, will remain energized. When the function receives power flow again, the current value again increments, beginning at the retained value. When reset R receives power flow, the current value is set back to zero and output Q is de-energized. On 350 and 360 series PLCs, if the enable to the ONDTR is low, PV = 0 and reset R receives power-flow, then the output will be low. However, on the 311–341 PLCs, under these same conditions, the output will be high.



Parameters:

Parameter	Description
address	The ONDTR uses three consecutive words (registers) of %R memory to store the following: • Current value (CV) = word 1. • Preset value (PV) = word 2. • Control word = word 3. When you enter an ONDTR, you must enter an address for the location of these three consecutive words (registers) directly below the graphic representing the function. **Note:** Do not use this address with other instructions. **Caution:** Overlapping references will result in erratic operation of the timer.
enable	When enable receives power flow, the timer's current value is incremented.
R	When R receives power flow, it resets the current value to zero.
PV	PV is the value to copy into the timer's preset value when the timer is enabled or reset.
Q	Output Q is energized when the current value is greater than or equal to the preset value.
time	Time increment is in tenths (0.1), hundredths (0.01), or thousandths (0.001) of seconds for the low bit of the PV preset value.

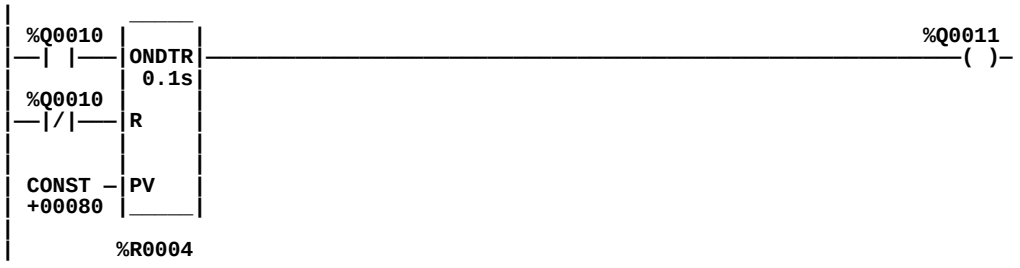
Valid Memory Types:

Parameter	flow	%I	%Q	%M	%T	%S	%G	%R	%AI	%AQ	const	none
address								•				
enable	•											
R	•											
PV		•	•	•	•		•	•	•	•	•	•
Q	•											•

• Valid reference or place where power may flow through the function.

Example:

In the following example, a retentive on-delay timer is used to create a signal (%Q0011) that turns on 8.0 seconds after %Q0010 turns on, and turns off when %Q0010 turns off.



TMR

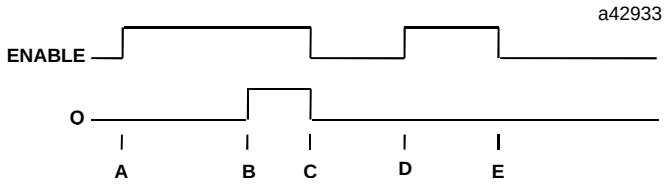
The simple on-delay timer (TMR) function increments while it receives power flow and resets to zero when power flow stops. Time may be counted in tenths of a second (the default selection), hundredths of a second, or thousandths of a second. The range is 0 to +32,767 time units. The state of this timer is retentive on power failure; no automatic initialization occurs at power-up.

When the TMR receives power flow, the timer starts accumulating time (current value). The current value is updated when it is encountered in the logic to reflect the total elapsed time the timer has been enabled since it was last reset.

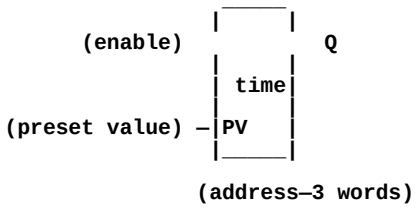
Note

If multiple occurrences of the same timer with the same reference address are enabled during a CPU sweep, the current values of the timers will be the same.

This update occurs as long as the enabling logic remains ON. When the current value equals or exceeds the preset value PV, the function begins passing power flow to the right. The timer continues accumulating time until the maximum value is reached. When the enabling parameter transitions from ON to OFF, the timer stops accumulating time and the current value is reset to zero.



A = ENABLE goes high; timer begins accumulating time.
B = Current value reaches preset value PV; Q goes high, and timer continues accumulating time.
C = ENABLE goes low; Q goes low; timer stops accumulating time and current time is cleared.
D = ENABLE goes high; timer starts accumulating time.
E = ENABLE goes low before current value reaches preset value PV; Q remains low; timer stops accumulating time and is cleared to zero.



Parameters:

Parameter	Description
address	The TMR uses three consecutive words (registers) of %R memory to store the following: • Current value (CV) = word 1. • Preset value (PV) = word 2. • Control word = word 3. When you enter a TMR, you must enter an address for the location of these three consecutive words (registers) directly below the graphic representing the function. **Note:** Do not use this address with other instructions. **Caution:** Overlapping references will result in erratic operation of the timer.
enable	When enable receives power flow, the timer's current value is incremented. When the TMR is not enabled, the current value is reset to zero and Q is turned off.
PV	PV is the value to copy into the timer's preset value when the timer is enabled or reset.
Q	Output Q is energized when TMR is enabled and the current value is greater than or equal to the preset value.

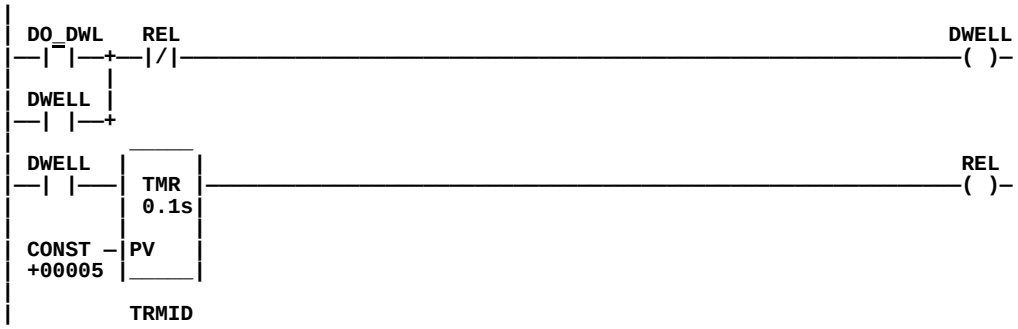
Valid Memory Types:

Parameter	flow	%I	%Q	%M	%T	%S	%G	%R	%AI	%AQ	const	none
address								•				
enable	•											
PV		•	•	•	•		•	•	•	•	•	•
Q	•											•

• Valid reference or place where power may flow through the function.

Example:

In the following example, a delay timer (with address) TMRID is used to control the length of time that coil DWELL is on. When the normally open (momentary) contact DO_DWL is on, coil DWELL is energized. The contact of coil DWELL keeps coil DWELL energized (when contact DO_DWL is released), and also starts the timer TMRID. When TMRID reaches its preset value of one-half second, coil REL energizes, interrupting the latched-on condition of coil DWELL. The contact DWELL interrupts power flow to TMRID, resetting its current value and de-energizing coil REL. The circuit is then ready for another momentary activation of contact DO_DWL.



OFDT

The off-delay timer (OFDT) increments while power flow is off, and resets to zero when power flow is on. Time may be counted in tenths of a second (the default selection), hundredths of a second, or thousandths of a second. The range is 0 to +32,767 time units. The state of this timer is retentive on power failure; no automatic initialization occurs at power-up.

When the OFDT first receives power flow, it passes power to the right, and the current value (CV) is set to zero. (The OFDT uses word 1 [register] as its CV storage location—see the "Parameters:" section on the next page for additional information.) The output remains on as long as the function receives power flow. If the function stops receiving power flow from the left, it continues to pass power to the right, and the timer starts accumulating time in the current value.

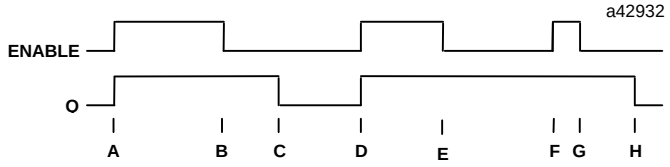
Note

If multiple occurrences of the same timer with the same reference address are enabled during a CPU sweep, the current values of the timers will be the same.

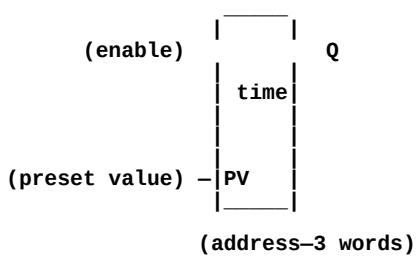
The OFDT does not pass power flow if the preset value is zero or negative.

Each time the function is invoked with the enabling logic set to OFF, the current value is updated to reflect the elapsed time since the timer was turned off. When the current value (CV) is equal to the preset value (PV), the function stops passing power flow to the right. When that occurs, the timer stops accumulating time—see Part C below.

When the function receives power flow again, the current value resets to zero.



A = ENABLE and Q both go high ; timer is reset (CV = 0).
B = ENABLE goes low; timer starts accumulating time.
C = CV reaches PV; Q goes low, and timer stops accumulating time.
D = ENABLE goes high; timer is reset (CV = 0).
E = ENABLE goes low; timer starts accumulating time.
F = ENABLE goes high; timer is reset (CV = 0).
G = ENABLE goes low; timer begins accumulating time.
H = CV reaches PV; Q goes low, and timer stops accumulating time.



When the OFDT is used in a program block that is ***not*** called every sweep, the timer accumulates time between calls to the program block unless it is reset. This means that it functions like a timer operating in a program with a much slower sweep than the timer in the main program block. For program blocks that are inactive for a long time, the timer should be programmed to allow for this catch-up feature. For example, if a timer in a program block is reset and the program block is not called (is inactive) for four minutes, when the program block is called, four minutes of time will already have accumulated. This time is applied to the timer when enabled, unless the timer is first reset.

Parameters:

Parameter	Description
address	The OFDT uses three consecutive words (registers) of %R memory to store the following: • Current value (CV) = word 1. • Preset value (PV) = word 2. • Control word = word 3. When you enter an OFDT, you must enter an address for the location of these three consecutive words (registers) directly below the graphic representing the function. **Note:** Do not use this address with other instructions. **Caution:** Overlapping references will result in erratic operation of the timer.
enable	When enable receives power flow, the timer's current value is incremented.
time	Time increment is in tenths (0.1), hundredths (0.01), or thousandths (0.001) of seconds for the low bit of the PV preset value.
PV	PV is the value to copy into the timer's preset value when the timer is enabled or reset.
Q	Output Q is energized when the current value is less than the preset value. The Q state is retentive on power failure; no automatic initialization occurs at power-up.

Valid Memory Types:

Parameter	flow	%I	%Q	%M	%T	%S	%G	%R	%AI	%AQ	const	none
address								•				
enable	•											
PV	•	•	•	•	•		•	•	•	•	•	•
Q	•											•

• Valid reference or place where power may flow through the function.

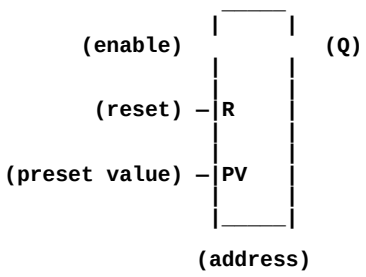
Example:

In the following example, an OFDT timer is used to turn off an output (%Q0001) whenever an input (%I0001) turns on. The output is turned on again 0.3 seconds after the input goes off.

UPCTR

The Up Counter (UPCTR) function is used to count up to a designated value. The range is 0 to +32,767 counts. When the up counter reset is ON, the current value of the counter is reset to 0. Each time the enable input transitions from OFF to ON, the current value is incremented by 1. The current value can be incremented past the preset value PV. The output is ON whenever the current value is greater than or equal to the preset value.

The state of the UPCTR is retentive on power failure; no automatic initialization occurs at power-up.



Parameters:

Parameter	Description
address	The UPCTR uses three consecutive words (registers) of %R memory to store the following: • Current value (CV) = word 1. • Preset value (PV) = word 2. • Control word = word 3. When you enter an UPCTR, you must enter an address for the location of these three consecutive words (registers) directly below the graphic representing the function. **Note:** Do not use this address with another up counter, down counter, or any other instruction or improper operation will result. **Caution:** Overlapping references will result in erratic operation of the counter.
enable	On a positive transition of enable, the current count is incremented by one.
R	When R receives power flow, it resets the current value back to zero.
PV	PV is the value to copy into the counter's preset value when the counter is enabled or reset.
Q	Output Q is energized when the current value is greater than or equal to the preset value.

Valid Memory Types:

Parameter	flow	%I	%Q	%M	%T	%S	%G	%R	%AI	%AQ	const	none
address								•				
enable	•											
R	•											
PV		•	•	•	•		•	•	•	•	•	•
Q	•											•

• Valid reference or place where power may flow through the function.

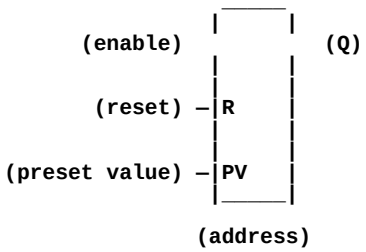
Example:

In the following example, every time input %I0012 transitions from OFF to ON, up counter PRT_CNT counts up by 1; internal coil %M0001 is energized whenever 100 parts have been counted. Whenever %M0001 is ON, the accumulated count is reset to zero.

DNCTR

The Down Counter (DNCTR) function is used to count down from a preset value. The minimum preset value is zero; the maximum present value is +32,767 counts. The minimum current value is –32,768. When reset, the current value of the counter is set to the preset value PV. When the enable input transitions from OFF to ON, the current value is decremented by one. The output is ON whenever the current value is less than or equal to zero.

The current value of the DNCTR is retentive on power failure; no automatic initialization occurs at power-up.



Parameters:

Parameter	Description
address	The DNCTR uses three consecutive words (registers) of %R memory to store the following: • Current value (CV)= word 1. • Preset value (PV) = word 2. • Control word = word 3. When you enter an DNCTR, you must enter an address for the location of these three consecutive words (registers) directly below the graphic representing the function. **Note:** Do not use this address with another down counter, up counter, or any other instruction or improper operation will result. **Caution:** Overlapping references will result in erratic operation of the counter.
enable	On a positive transition of enable, the current value is decremented by one.
R	When R receives power flow, it resets the current value to the preset value.
PV	PV is the value to copy into the counter's preset value when the counter is enabled or reset.
Q	Output Q is energized when the current value is less than or equal to zero.

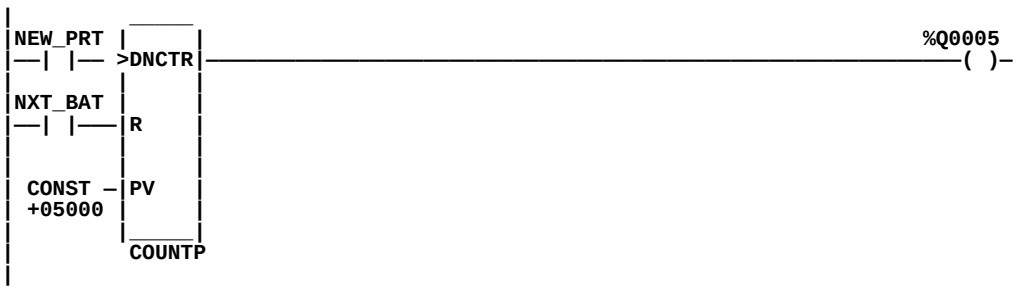
Valid Memory Types:

Parameter	flow	%I	%Q	%M	%T	%S	%G	%R	%AI	%AQ	const	none
address								•				
enable	•											
R	•											
PV		•	•	•	•		•	•	•	•	•	•
Q	•											•

• Valid reference or place where power may flow through the function.

Example:

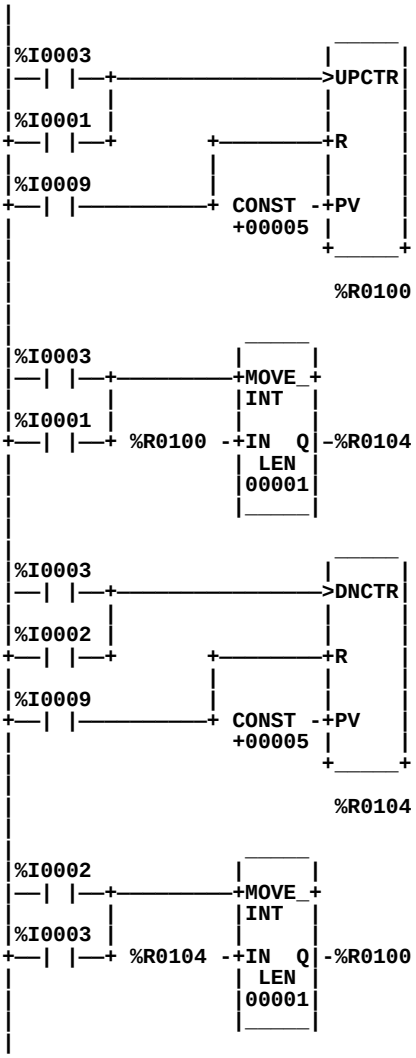
In the following example, the down counter identified as COUNTP counts 5000 new parts before energizing output %Q0005.



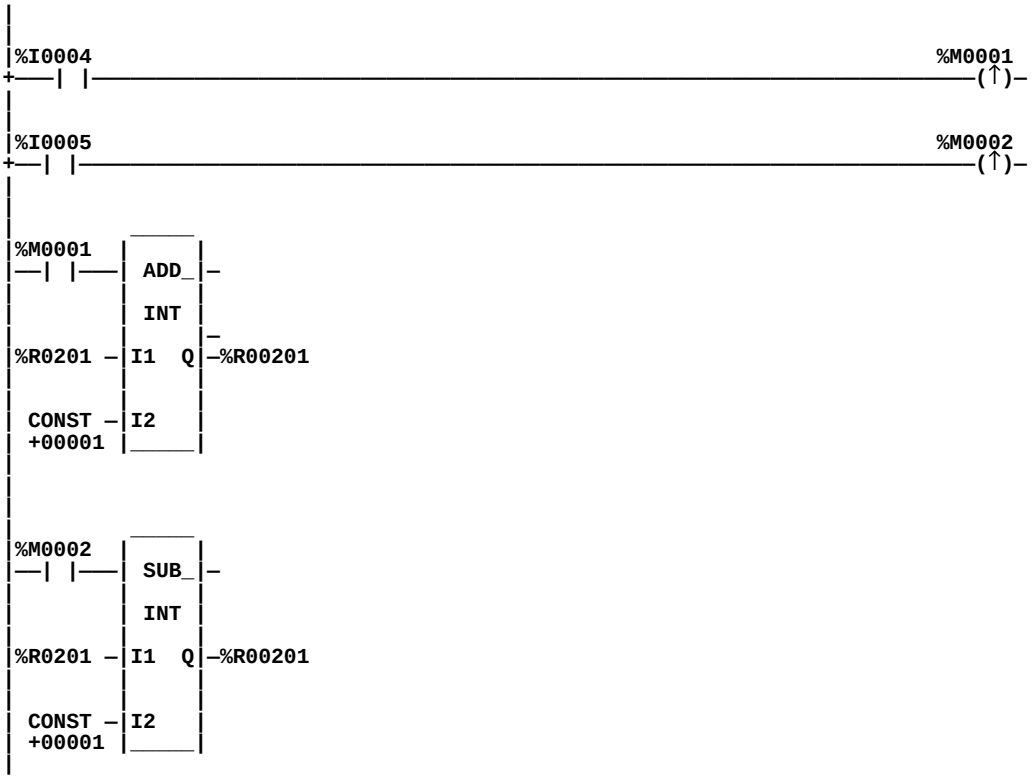
Example:

In the following example, the PLC is used to keep track of the number of parts contained in a temporary storage area. There are two ways of accomplishing this function using the Series 90-30/20/Micro instruction set.

The first method is to use an up/down counter pair with a shared register for the accumulated or current value. When the parts enter the storage area, the up counter increments by 1, increasing the current value of the parts in storage by a value of 1. When a part leaves the storage area, the down counter decrements by 1, decreasing the inventory storage value by 1. To avoid conflict with the shared register, both counters use different register addresses. When a register counts, its current value must be moved to the current value register of the other counter.



The second method, shown below, uses the ADD and SUB functions to provide storage tracking.



Section 3: Math Functions

This section describes the math functions of the Series 90-30/20/Micro Instruction Set:

Abbreviation	Function	Description	Page
ADD	Addition	Add two numbers.	4-27
SUB	Subtraction	Subtract one number from another.	4-27
MUL	Multiplication	Multiply two numbers.	4-27
DIV	Division	Divide one number by another, yielding a quotient.	4-27
MOD	Modulo Division	Divide one number by another, yielding a remainder.	4-31
SQRT	Square Root	Find the square root of an integer or real value.	4-33
SIN, COS, TAN, ASIN, ACOS, ATAN	Trigonometric Functions †	Perform the appropriate function on the real value in input IN.	4-35
LOG, LN EXP, EXPT	Logarithmic/Exponential Functions †	Perform the appropriate function on the real value in input IN.	4-37
RAD, DEG	Radian Conversion †	Perform the appropriate function on the real value in input IN.	4-39

† Trigonometric Functions, Logarithmic/Exponential Functions, and Radian Conversion functions are only available on the model 350 and 360 series CPUs, Release 9 or later, or on all releases of CPU352.

Note

Division and modulo division are similar functions which differ in their output; division finds a quotient, while modulo division finds a remainder.

Standard Math Functions (ADD, SUB, MUL, DIV)

Math functions include addition, subtraction, multiplication, and division. When a function receives power flow, the appropriate math function is performed on input parameters I1 and I2. These parameters must be the same data type. Output Q is the same data type as I1 and I2.

Note

DIV rounds down; it does not round to the closest integer.
(For example, 24 DIV 5 = 4.)

Math functions operate on these types of data:

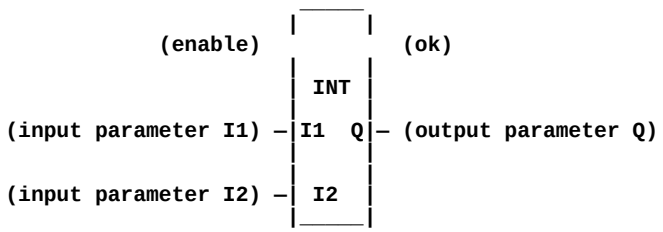
Data Type	Description
INT	Signed integer.
DINT	Double precision signed integer.
REAL	Floating Point

Note

The REAL data type is only available on 350 and 360 series CPUs, Release 9 or later, or on all releases of CPU352.

The default data type is signed integer; however, it can be changed after selecting the function. For more information on data types, please refer to chapter 2, section 2, “Program Organization and User References/Data.”

If the operation of INT or DINT results in overflow, the output reference is set to its largest possible value for the data type. For signed numbers, the sign is set to show the direction of the overflow. If the operation does not result in overflow (and the inputs are valid numbers), the ok output is set ON; otherwise, it is set OFF. If signed or double precision integers are used, the sign of the result for DIV and MUL functions depends on the signs of I1 and I2.



Parameters:

Parameter	Description
enable	When the function is enabled, the operation is performed.
I1	I1 contains a constant or reference for the first value used in the operation. (I1 is on the left side of the mathematical equation, as in I1 — I2).
I2	I2 contains a constant or reference for the second value used in the operation. (I2 is on the right side of the mathematical equation, as in I1 — I2).
ok	The ok output is energized when the function is performed without overflow, unless an invalid operation occurs.
Q	Output Q contains the result of the operation.

Valid Memory Types:

Parameter	flow	%I	%Q	%M	%T	%S	%G	%R	%AI	%AQ	const	none
enable	•											
I1		o	o	o	o		o	•	•	•	•†	
I2		o	o	o	o		o	•	•	•	•†	
ok	•											•
Q		o	o	o	o		o	•	•	•		

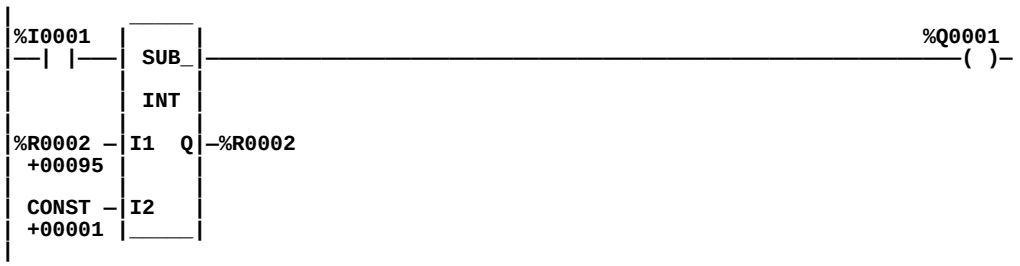
• Valid reference or place where power may flow through the function.
o Valid reference for INT data only; not valid for DINT or REAL.
† Constants are limited to values between –32,768 and +32,767 for double precision signed integer operations.

Note

The default type is INT for 16-bit or single register operands. Press **F10** to change the Types selection to DINT, 32-bit double word, or REAL (for the 350 and 360 series CPUs only). PLC INT values occupy a single 16-bit register, %R, %AI or %AQ. DINT values require two consecutive registers with the low 16 bits in the first word and the upper 16 bits with the sign in second word. REAL values, in the 350 and 360 series CPUs only—Release 9 or later, plus all releases of CPU352, also occupy a 32-bit double register with the sign in the high bit followed by the exponent and mantissa.

Example:

In the following example, whenever input %I0001 is set, the integer content of %R0002 is decremented by 1 and coil %Q0001 is turned on, provided there is no overflow in the subtraction.



Math Functions and Data Types

Function	Operation	Displays as
ADD INT	Q(16 bit) = I1(16 bit) + I2(16 bit)	5-digit base 10 number with sign
ADD DINT	Q(32 bit) = I1(32 bit) + I2(32 bit)	8-digit base 10 number with sign
ADD REAL*	Q(32 bit) = I1(32 bit) + I2(32 bit)	7-digit base 10 number, sign and decimal
SUB INT	Q(16 bit) = I1(16 bit) – I2(16 bit)	5-digit base 10 number with sign
SUB DINT	Q(32 bit) = I1(32 bit) – I2(32 bit)	8-digit base 10 number with sign
SUB REAL*	Q(32 bit) = I1(32 bit) – I2(32 bit)	7-digit base 10 number, sign and decimal
MUL INT	Q(16 bit) = I1(16 bit) * I2(16 bit)	5-digit base 10 number with sign
MUL DINT	Q(32 bit) = I1(32 bit) * I2(32 bit)	8-digit base 10 number with sign
MUL REAL*	Q(32 bit) = I1(32 bit) * I2(32 bit)	7-digit base 10 number, sign and decimal
DIV INT	Q(16 bit) = I1(16 bit) / I2(16 bit)	5-digit base 10 number with sign
DIV DINT	Q(32 bit) = I1(32 bit) / I2(32 bit)	8-digit base 10 number with sign
DIV REAL*	Q(32 bit) = I1(32 bit) / I2(32 bit)	7-digit base 10 number, sign and decimal

* 350 and 360 series CPUs only, Release 9 or later, or all releases of CPU352

Note

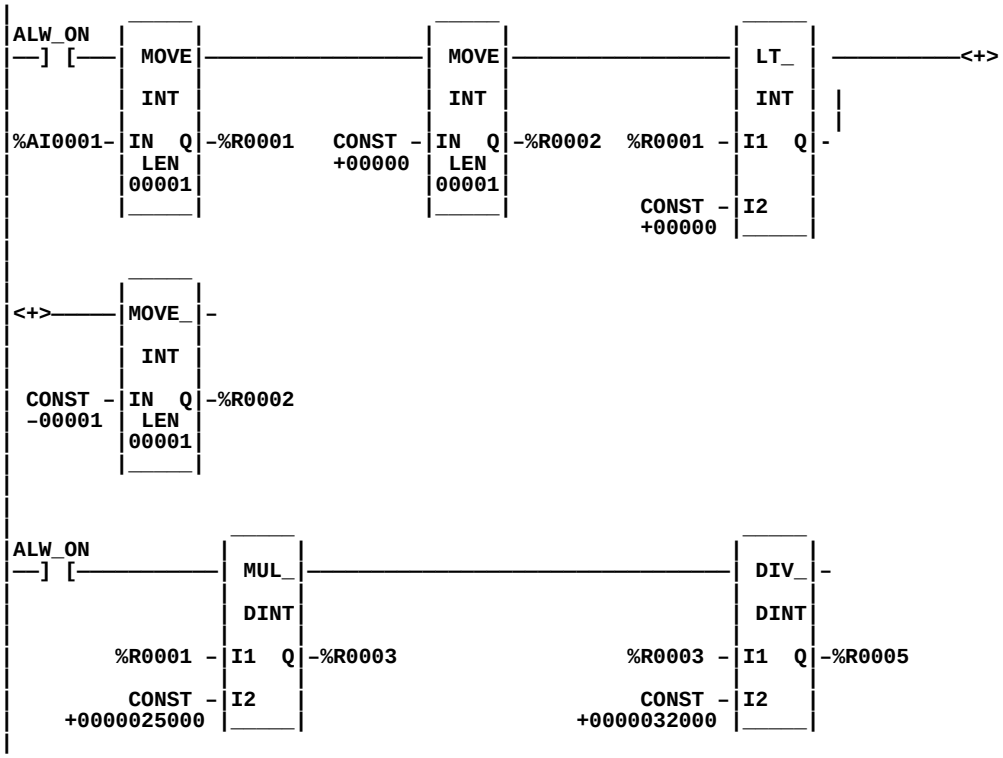
The input and output data types must be the same. The MUL and DIV functions do not support a mixed mode as the 90-70 PLCs do. For example, the MUL INT of 2 16-bit inputs produces a 16-bit product, not a 32-bit product. Using MUL DINT for a 32-bit product requires both inputs to be 32-bit. The DIV INT divides a 16-bit I2 for a 16-bit result while DIV DINT divides a 32-bit I1 by 32-bit I2 for a 32-bit result.

These functions pass power if there is no math overflow. If an overflow occurs, the result is the largest value with the proper sign and no power flow.

Be careful to avoid overflows when using MUL and DIV functions. If you have to convert INT to DINT values, remember that the CPU uses standard 2's complement with the sign extended to the highest bit of the second word. You must check the sign of the low 16-bit word and extend it into the second 16 bit word. If the most significant bit in a 16-bit INT word is 0 (positive), move a 0 to the second word. If the most significant bit in a 16-bit word is –1 (negative), move a –1 or hex 0FFFFh to the second word. Converting from DINT to INT is easier as the low 16-bit word (first register) is the INT part of a DINT 32-bit word. The upper 16 bits or second word should be either a 0 (positive) or –1 (negative) value or the DINT number is too big to convert to 16 bit.

A common application is to scale analog input values with a MUL operation followed by a DIV and possibly an ADD operation. With a range up to 32000, using a MUL INT will overflow. Using an %AI value for a MUL DINT will also not work as the 32-bit I1 will combine 2 analog inputs at the same time. You must move the analog input to the low word of a double register, then test the sign and set the second register to 0 if positive or –1 if it was negative. Use the double register with the MUL DINT for a 32 product for the following DIV function.

For example, the following logic could be used to scale a +/–10 volt input %AI1 to +/– 25000 engineering units in %R5.



MOD (INT, DINT)

The Modulo (MOD) function is used to divide one value by another value of the same data type, to obtain the remainder. The sign of the result is always the same as the sign of input parameter I1.

The MOD function operates on these types of data:

Data Type	Description
INT	Signed integer.
DINT	Double precision signed integer.

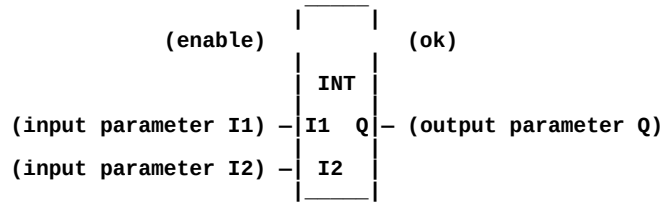
The default data type is signed integer; however, it can be changed after selecting the function. For more information on data types, please refer to chapter 2, section 2, "Program Organization and User References/Data."

When the function receives power flow, it divides input parameter I1 by input parameter I2. These parameters must be the same data type. Output Q is calculated using the formula:

```
Q = I1 - ((I1 DIV I2) * I2)
```

where DIV produces an integer number. Q is the same data type as input parameters I1 and I2.

OK is always ON when the function receives power flow, unless there is an attempt to divide by zero. In that case, it is set OFF.



Parameters:

Parameter	Description
enable	When the function is enabled, the operation is performed.
I1	I1 contains a constant or reference for the value to be divided by I2.
I2	I2 contains a constant or reference for the value to be divided into I1.
ok	The ok output is energized when the function is performed without overflow.
Q	Output Q contains the result of dividing I1 by I2 to obtain a remainder.

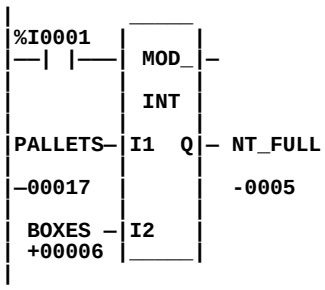
Valid Memory Types:

Parameter	flow	%I	%Q	%M	%T	%S	%G	%R	%AI	%AQ	const	none
enable	•											
I1		o	o	o	o		o	•	•	•	•†	
I2		o	o	o	o		o	•	•	•	•†	
ok	•											•
Q		o	o	o	o		o	•	•	•		

• Valid reference or place where power may flow through the function.
o Valid reference for INT data only; not valid for DINT.
† Constants are limited to values between –32,768 and +32,767 for double precision signed integer operations.

Example:

In the following example, the remainder of the integer division of BOXES into PALLETS is placed into NT_FULL whenever %I0001 is ON.



SQRT (INT, DINT, REAL)

The Square Root (SQRT) function is used to find the square root of a value. When the function receives power flow, the value of output Q is set to the integer portion of the square root of the input IN. The output Q must be the same data type as IN.

The SQRT function operates on these types of data:

Data Type	Description
INT	Signed integer.
DINT	Double precision signed integer.
REAL	Floating Point.

Note

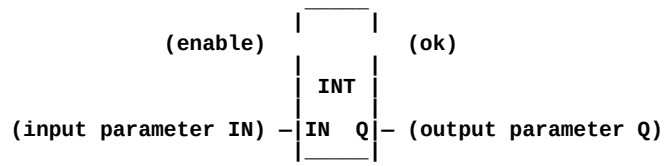
The REAL data type is only available on 350 and 360 series CPUs, Release 9 or later, or on all releases of CPU352.

The default data type is signed integer; however, it can be changed after selecting the function. For more information on data types, please refer to chapter 2, section 2, "Program Organization and User References/Data."

OK is set ON if the function is performed without overflow, unless one of these invalid REAL operations occurs:

- IN < 0.
- IN is NaN (Not a Number).

Otherwise, ok is set OFF.



Parameters:

Parameter	Description
enable	When the function is enabled, the operation is performed.
IN	IN contains a constant or reference for the value whose square root is to be calculated. If IN is less than zero, the function will not pass power flow.
ok	The ok output is energized when the function is performed without overflow, unless an invalid operation occurs.
Q	Output Q contains the square root of IN.

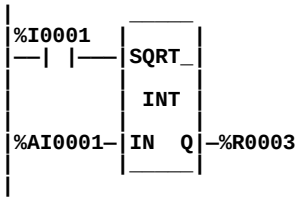
Valid Memory Types:

Parameter	flow	%I	%Q	%M	%T	%S	%G	%R	%AI	%AQ	const	none
enable	•											
IN		o	o	o	o		o	•	•	•	•†	
ok	•											•
Q		o	o	o	o		o	•	•	•		

• Valid reference or place where power may flow through the function.
o Valid reference for INT data only; not valid for DINT and REAL.
† Constants are limited to values between –32,768 and +32,767 for double precision signed integer operations.

Example:

In the following example, the square root of the integer number located at %AI001 is placed into the result located at %R0003 whenever %I0001 is ON.



Trig Functions (SIN, COS, TAN, ASIN, ACOS, ATAN)

The SIN, COS, and TAN functions are used to find the trigonometric sine, cosine, and tangent, respectively, of its input. When one of these functions receives power flow, it computes the sine (or cosine or tangent) of IN, whose units are radians, and stores the result in output Q. Both IN and Q are floating-point values.

The ASIN, ACOS, and ATAN functions are used to find the inverse sine, cosine, and tangent, respectively, of its input. When one of these functions receives power flow, it computes the inverse sine (or cosine or tangent) of IN and stores the result in output Q, whose units are radians. Both IN and Q are floating-point values.

The SIN, COS, and TAN functions accept a broad range of input values, where $-2^{63} < IN < +2^{63}$, ($2^{63} \approx 9.22 \times 10^{18}$).

The ASIN and ACOS functions accept a narrow range of input values, where $-1 \leq IN \leq 1$. Given a valid value for the IN parameter, the ASIN_REAL function will produce a result Q such that:

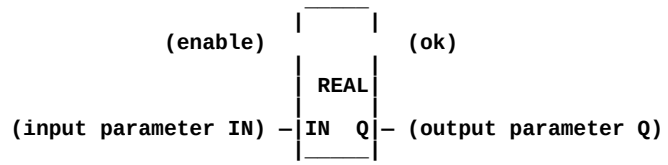
$$\mathbf{ASIN\ (IN)} = -\frac{\pi}{2} \leq Q \leq \frac{\pi}{2}$$

The ACOS_REAL function will produce a result Q such that:

$$\mathbf{ACOS\ (IN)} = 0 \leq Q \leq \pi$$

The ATAN function accepts the broadest range of input values, where $-\infty \leq IN \leq +\infty$. Given a valid value for the IN parameter, the ATAN_REAL function will produce a result Q such that:

$$\mathbf{ATAN\ (IN)} = -\frac{\pi}{2} \leq Q \leq \frac{\pi}{2}$$



Note

The TRIG functions are only available on the 350 and 360 series CPUs, Release 9 or later, or on all releases of CPU352.

Parameters:

Parameter	Description
enable	When the function is enabled, the operation is performed.
IN	IN contains the constant or reference real value to be operated on.
ok	The ok output is energized when the function is performed without overflow, unless an invalid operation occurs and/or IN is NaN.
Q	Output Q contains the trigonometric value of IN.

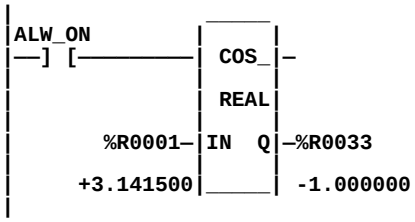
Valid Memory Types:

Parameter	flow	%I	%Q	%M	%T	%S	%G	%R	%AI	%AQ	const	none
enable	•											
IN								•	•	•	•	
ok	•											•
Q								•	•	•		

• Valid reference or place where power may flow through the function.

Example:

In the following example, the COS of the value in %R0001 is placed in %R0033.

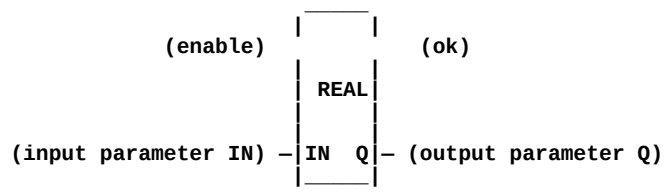


Logarithmic/Exponential Functions (LOG, LN, EXP, EXPT)

The LOG, LN, and EXP functions have two input parameters and two output parameters. When the function receives power flow, it performs the appropriate logarithmic/exponential operation on the real value in input IN and places the result in output Q.

- For the LOG function, the base 10 logarithm of IN is placed in Q.
- For the LN function, the natural logarithm of IN is placed in Q.
- For the EXP function, ***e*** is raised to the power specified by IN and the result is placed in Q.
- For the EXPT function, the value of input I1 is raised to the power specified by the value I2 and the result is placed in output Q. (The EXPT function has three input parameters and two output parameters.)

The ok output will receive power flow, unless IN is NaN (Not a Number) or is negative.



Parameters:

Parameter	Description
enable	When the function is enabled, the operation is performed.
IN	IN contains the real value to be operated on.
ok	The ok output is energized when the function is performed without overflow, unless an invalid operation occurs and/or IN is NaN or is negative.
Q	Output Q contains the logarithmic/exponential value of IN.

Note

The LOG, LN, EXP and EXPT functions are only available on the 350 and 360 series CPUs, Release 9 or later, or on all releases of CPU352.

Valid Memory Types:

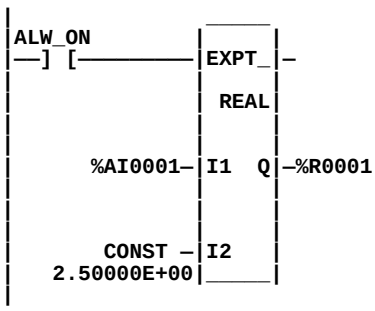
Parameter	flow	%I	%Q	%M	%T	%S	%G	%R	%AI	%AQ	const	none
enable	•											
IN*								•	•	•	•	
ok	•											•
Q								•	•	•		

* For the EXPT function, input IN is replaced by input parameters I1 and I2.

• Valid reference or place where power may flow through the function.

Example:

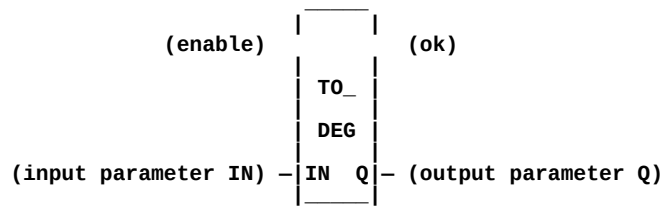
In the following example, the value of %AI0001 is raised to the power of 2.5 and the result is placed in %R0001.



Radian Conversion (RAD, DEG)

When the function receives power flow, the appropriate conversion (RAD_TO_DEG or DEG_TO_RAD, i.e., Radian to Degree or vice versa) is performed on the real value in input IN and the result is placed in output Q.

The ok output will receive power flow unless IN is NaN (Not a Number).



Parameters:

Parameter	Description
enable	When the function is enabled, the operation is performed.
IN	IN contains the real value to be operated on.
ok	The ok output is energized when the function is performed without overflow, unless IN is NaN.
Q	Output Q contains the converted value of IN.

Note

The Radian conversion functions are only available on the 350 and 360 series CPUs, Release 9 or later, or on all releases of CPU352.

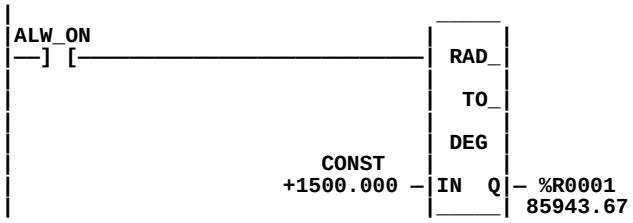
Valid Memory Types:

Parameter	flow	%I	%Q	%M	%T	%S	%G	%R	%AI	%AQ	const	none
enable	•											
IN								•	•	•	•	
ok	•											•
Q								•	•	•		

• Valid reference or place where power may flow through the function.

Example:

In the following example, +1500 is converted to DEG and is placed in %R0001.



Section 4: Relational Functions

Relational functions are used to compare two numbers. This section describes the following relational functions:

Abbreviation	Function	Description	Page
EQ	Equal	Test two numbers for equality.	4-41
NE	Not Equal	Test two numbers for non-equality.	4-41
GT	Greater Than	Test for one number greater than another.	4-41
GE	Greater Than or Equal	Test for one number greater than or equal to another.	4-41
LT	Less Than	Test for one number less than another.	4-41
LE	Less Than or Equal	Test for one number less than or equal to another.	4-41
RANGE	Range	Determine whether a number is within a specified range (available for Release 4.5 or higher CPUs).	4-44

Relational functions are used to determine the relation of two values. When the function receives power flow, it compares input parameter I1 to input parameter I2. These parameters must be the same data type. Relational functions operate on these types of data:

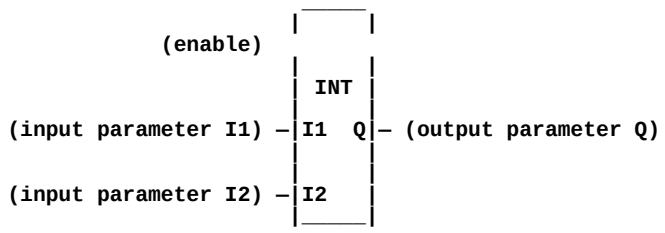
Data Type	Description
INT	Signed integer.
DINT	Double precision signed integer.
REAL	Floating Point

Note

The REAL data type is only available on the 350 and 360 series CPUs, Release 9 or later, or on all releases of CPU352. Also, the Range function block does not accept REAL type. Additionally, the %S0020 bit is set ON when a relational function using REAL data executes successfully. It is cleared when either input is NaN (Not a Number).

The default data type is signed integer. To compare either signed integers, double precision signed integers, or real numbers select the new data type after selecting the relational function. To compare data of other types or of two different types, first use the appropriate conversion function (described in section 8, “Conversion Functions”) to change the data to one of the supported types.

If input parameters I1 and I2 match the specified relation, output Q receives power flow and is set ON (1); otherwise, it is set OFF (0).



Parameters:

Parameter	Description
enable	When the function is enabled, the operation is performed.
I1	I1 contains a constant or reference for the first value to be compared. (I1 is on the left side of the relational equation, as in I1 < I2).
I2	I2 contains a constant or reference for the second value to be compared. (I2 is on the right side of the relational equation, as in I1 < I2).
Q	Output Q is energized when I1 and I2 match the specified relation.

Note

I1 and I2 must be valid numbers, i.e., cannot be NaN (Not a Number).

Expanded Description:

Function	Description
Equal	When enabled, if the value at input I1 is equal to the value at input I2, output Q is energized.
Not Equal	When enabled, if the value at input I1 is NOT equal to the value at input I2, output Q is energized.
Greater Than	When enabled, if the value at input I1 is greater than the value at input I2, output Q is energized.
Greater Than or Equal	When enabled, if the value at input I1 is greater than or equal to the value at input I2, output Q is energized.
Less Than	When enabled, if the value at input I1 is less than the value at input I2, output Q is energized.
Less Than or Equal	When enabled, if the value at input I1 is less than or equal to the value at input I2, output Q is energized.

Valid Memory Types:

Parameter	flow	%I	%Q	%M	%T	%S	%G	%R	%AI	%AQ	const	none
enable	•											
I1		o	o	o	o		o	•	•	•	•†	
I2		o	o	o	o		o	•	•	•	•†	
Q	•											•

• Valid reference or place where power may flow through the function.
o Valid reference for INT data only; not valid for DINT or REAL.
† Constants are limited to integer values for double precision signed integer operations.

Example:

In the following example, two double precision signed integers, PWR_MDE and BIN_FUL, are compared whenever %I0001 is set. If PWR_MDE is less than or equal to BIN_FUL, coil %Q0002 is turned on.

RANGE (INT, DINT, WORD)

The RANGE function is used to determine if a value is between the range of two numbers.

Note

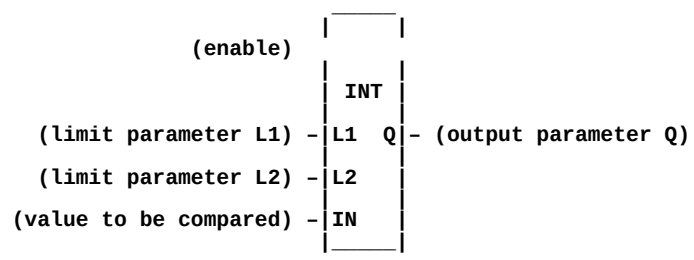
This function is available *only* to Release 4.41 or later CPUs.

The RANGE function operates on these types of data:

Data Type	Description
INT	Signed integer.
DINT	Double precision signed integer.
WORD	Word data type.

The default data type is signed integer; however, it can be changed after selecting the function. For more information on data types, please refer to chapter 2, section 2, "Program Organization and User References/Data."

When the function is enabled, the RANGE function block will compare the value in input parameter IN against the range specified by limit parameters L1 and L2. When the value is within the range specified by L1 and L2, inclusive, output parameter Q is set ON (1). Otherwise, Q is set OFF (0).



Note

Limit parameters L1 and L2 represent the end points of a range. There is no minimum/maximum or high/low connotation assigned to either parameter. Thus, a desired range of 0 to 100 could be specified by assigning 0 to L1 and 100 to L2 or 0 to L2 and 100 to L1.

Parameters:

Parameter	Description
enable	When the function is enabled, the operation is performed.
L1	L1 contains the start point of the range.
L2	L2 contains the end point of the range.
IN	IN contains the value to be compared against the range specified by L1 and L2.
Q	Output Q is energized when the value in IN is within the range specified by L1 and L2, inclusive.

Valid Memory Types:

Parameter	flow	%I	%Q	%M	%T	%S	%G	%R	%AI	%AQ	const	none
enable	•											
L1		o	o	o	o		o	•	•	•	•‡	
L2		o	o	o	o		o	•	•	•	•‡	
IN		o	o	o	o		o	•	•	•		
Q	•											•

• Valid reference or place where power may flow through the function.
o Valid reference for INT or WORD data only; not valid for DINT.
‡ Constants are limited to integer values for double precision signed integer operations.

Example 1:

In the following example, %AI0001 is checked to be within a range specified by two constants, 0 and 100.

RANGE Truth Table				
Enable State %I0001	**L1 Value Constant**	**L2 Value Constant**	**IN Value %AI0001**	**Q State %Q0001**
ON	100	0	< 0	OFF
ON	100	0	0 — 100	ON
ON	100	0	> 100	OFF
OFF	100	0	Not Applicable	OFF

Example 2:

In this example, %AI0001 is checked to be within a range specified by two register values.

RANGE Truth Table				
Enable State %I0001	**L1 Value %R0001**	**L2 Value %R0002**	**IN Value %AI0001**	**Q State %Q0001**
ON	500	0	< 0	OFF
ON	500	0	0 — 500	ON
ON	500	0	> 500	OFF
OFF	500	0	Not Applicable	OFF

Section 5: Bit Operation Functions

Bit operation functions perform comparison, logical, and move operations on bit strings. The AND, OR, XOR, and NOT functions operate on a single word. The remaining bit operation functions may operate on multiple words, with a maximum string length of 256 words. All bit operation functions require WORD data.

Although data must be specified in 16-bit increments, these functions operate on data as a continuous string of bits, with bit 1 of the first word being the Least Significant Bit (LSB). The last bit of the last word is the Most Significant Bit (MSB). For example, if you specified three words of data beginning at reference %R0100, it would be operated on as 48 contiguous bits.

%R0100	**16**	**15**	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1** ← **bit 1 (LSB)**
%R0101	**32**	**31**	**30**	**29**	**28**	**27**	**26**	**25**	**24**	**23**	**22**	**21**	**20**	**19**	**18**	**17**
%R0102	**48**	**47**	**46**	**45**	**44**	**43**	**42**	**41**	**40**	**39**	**38**	**37**	**36**	**35**	**34**	**33**

↑

(MSB)

Note

Overlapping input and output reference address ranges in multi-word functions may produce unexpected results.

The following bit operation functions are described in this section:

Abbreviation	Function	Description	Page
AND	Logical AND	If a bit in bit string I1 and the corresponding bit in bit string I2 are both 1, place a 1 in the corresponding location in output string Q.	4-49
OR	Logical OR	If a bit in bit string I1 and/or the corresponding bit in bit string I2 are both 1, place a 1 in the corresponding location in output string Q.	4-49
XOR	Logical exclusive OR	If a bit in bit string I1 and the corresponding bit in string I2 are different, place a 1 in the corresponding location in the output bit string.	4-51
NOT	Logical invert	Set the state of each bit in output bit string Q to the opposite state of the corresponding bit in bit string I1.	4-53
SHL	Shift Left	Shift all the bits in a word or string of words to the left by a specified number of places.	4-55
SHR	Shift Right	Shift all the bits in a word or string of words to the right by a specified number of places.	4-55
ROL	Rotate Left	Rotate all the bits in a string a specified number of places to the left.	4-58
ROR	Rotate Right	Rotate all the bits in a string a specified number of places to the right.	4-57
BTST	Bit Test	Test a bit within a bit string to determine whether that bit is currently 1 or 0.	4-60
BSET	Bit Set	Set a bit in a bit string to 1.	4-62
BCLR	Bit Clear	Clear a bit within a string by setting that bit to 0.	4-62
BPOS	Bit Position	Locate a bit set to 1 in a bit string.	4-64
MSKCMP	Masked Compare	Compare the contents of two separate bit strings with the ability to mask selected bits (available for Release 4.5 or higher CPUs).	4-66

AND and OR (WORD)

Each scan that power is received, the AND or OR function examines each bit in bit string I1 and the corresponding bit in bit string I2, beginning at the least significant bit in each.

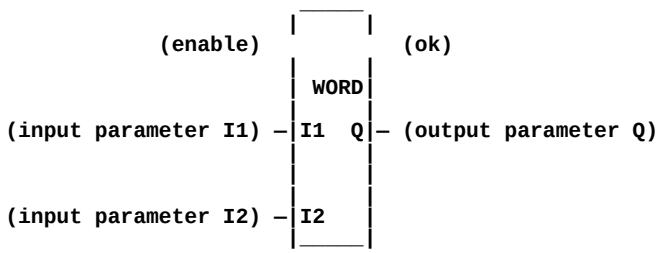
For each two bits examined for the AND function, if both are 1, then a 1 is placed in the corresponding location in output string Q. If either or both bits are 0, then a 0 is placed in string Q in that location.

The AND function is useful for building masks or screens, where only certain bits are passed through (those that are opposite a 1 in the mask), and all other bits are set to 0. The function can also be used to clear the selected area of word memory by ANDing the bits with another bit string known to contain all 0s. The I1 and I2 bit strings specified may overlap.

For each two bits examined for the OR function, if either or both bits are 1, then a 1 is placed in the corresponding location in output string Q. If both bits are 0, then a 0 is placed in string Q in that location.

The OR function is useful for combining strings, and to control many outputs through the use of one simple logical structure. The function is the equivalent of two relay contacts in parallel multiplied by the number of bits in the string. It can be used to drive indicator lamps directly from input states, or superimpose blinking conditions on status lights.

The function passes power flow to the right whenever power is received.



Parameters:

Parameter	Description
enable	When the function is enabled, the operation is performed.
I1	I1 contains a constant or reference for the first word of the first string.
I2	I2 contains a constant or reference for the first word of the second string.
ok	The ok output is energized whenever enable is energized.
Q	Output Q contains the result of the operation.

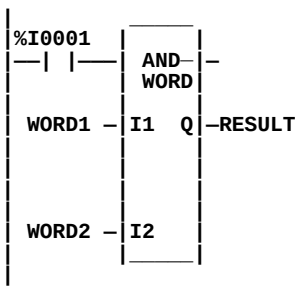
Valid Memory Types:

Parameter	flow	%I	%Q	%M	%T	%S	%G	%R	%AI	%AQ	const	none
enable	•											
I1		•	•	•	•	•	•	•	•	•	•	
I2		•	•	•	•	•	•	•	•	•	•	
ok	•											•
Q		•	•	•	•	•†	•	•	•	•		

• Valid reference or place where power may flow through the function.
† %SA, %SB, or %SC only; %S cannot be used.

Example:

In the following example, whenever input %I0001 is set, the 16-bit strings represented by nicknames WORD1 and WORD2 are examined. The results of the Logical AND are placed in output string RESULT.



WORD1	**0**	**0**	**0**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**0**	**0**	**1**	**0**	**0**	**0**
WORD2	**1**	**1**	**0**	**1**	**1**	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**1**	**1**	**1**	**1**

RESULT	**0**	**0**	**0**	**1**	**1**	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**1**	**0**	**0**	**0**

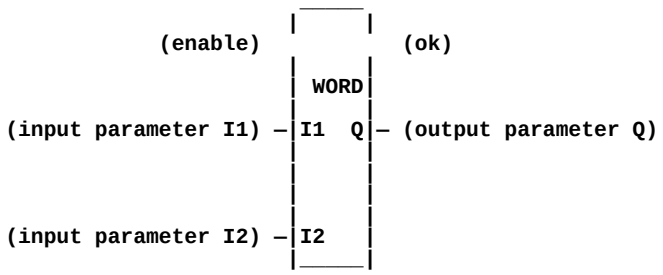
XOR (WORD)

The Exclusive OR (XOR) function is used to compare each bit in bit string I1 with the corresponding bit in string I2. If the bits are different, a 1 is placed in the corresponding position in the output bit string.

Each scan that power is received, the function examines each bit in string I1 and the corresponding bit in string I2, beginning at the least significant bit in each. For each two bits examined, if only one is 1, then a 1 is placed in the corresponding location in bit string Q. The XOR function passes power flow to the right whenever power is received.

If string I2 and output string Q begin at the same reference, a 1 placed in string I1 will cause the corresponding bit in string I2 to alternate between 0 and 1, changing state with each scan as long as power is received. Longer cycles can be programmed by pulsing the power flow to the function at twice the desired rate of flashing; the power flow pulse should be one scan long (one-shot type coil or self-resetting timer).

The XOR function is useful for quickly comparing two bit strings, or to blink a group of bits at the rate of one ON state per two scans.



Parameters:

Parameter	Description
enable	When the function is enabled, the operation is performed.
I1	I1 contains a constant or reference for the first word to be XORed.
I2	I2 contains a constant or reference for the second word to be XORed.
ok	The ok output is energized whenever enable is energized.
Q	Output Q contains the result of I1 XORed with I2.

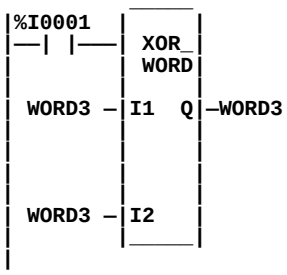
Valid Memory Types:

Parameter	flow	%I	%Q	%M	%T	%S	%G	%R	%AI	%AQ	const	none
enable	•											
I1		•	•	•	•	•	•	•	•	•	•	
I2		•	•	•	•	•	•	•	•	•	•	
ok	•											•
Q		•	•	•	•	•†	•	•	•	•		

• Valid reference or place where power may flow through the function.
† %SA, %SB, or %SC only; %S cannot be used.

Example:

In the following example, whenever %I0001 is set, the bit string represented by the nickname WORD3 is cleared (set to all zeros).



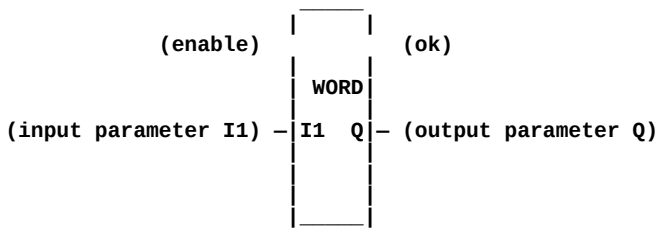
I1 (WORD3)	0	0	0	1	1	1	1	1	1	1	0	0	1	0	0	0
I2 (WORD3)	0	0	0	1	1	1	1	1	1	1	0	0	1	0	0	0

Q (WORD3)	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

NOT (WORD)

The NOT function is used to set the state of each bit in the output bit string Q to the opposite of the state of the corresponding bit in bit string I1.

All bits are altered on each scan that power is received, making output string Q the logical complement of I1. The function passes power flow to the right whenever power is received.



Parameters:

Parameter	Description
enable	When the function is enabled, the operation is performed.
I1	I1 contains the constant or reference for the word to be negated.
ok	The ok output is energized whenever enable is energized.
Q	Output Q contains the NOT (negation) of I1.

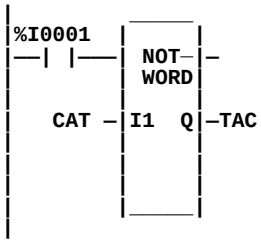
Valid Memory Types:

Parameter	flow	%I	%Q	%M	%T	%S	%G	%R	%AI	%AQ	const	none
enable	•											
I1		•	•	•	•	•	•	•	•	•	•	
ok	•											•
Q		•	•	•	•	•†	•	•	•	•		

• Valid reference or place where power may flow through the function.
† %SA, %SB, or %SC only; %S cannot be used.

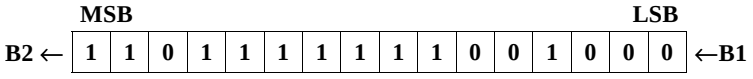
Example:

In the following example, whenever input %I0001 is set, the bit string represented by the nickname TAC is set to the inverse of bit string CAT.

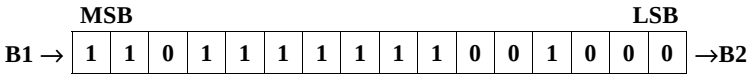


SHL and SHR (WORD)

The Shift Left (SHL) function is used to shift all the bits in a word or group of words to the left by a specified number of places. When the shift occurs, the specified number of bits is shifted out of the output string to the left. As bits are shifted out of the high end of the string, the same number of bits is shifted in at the low end.



The Shift Right (SHR) function is used to shift all the bits in a word or group of words a specified number of places to the right. When the shift occurs, the specified number of bits is shifted out of the output string to the right. As bits are shifted out of the low end of the string, the same number of bits is shifted in at the high end.



A string length of 1 to 256 words can be selected for either function.

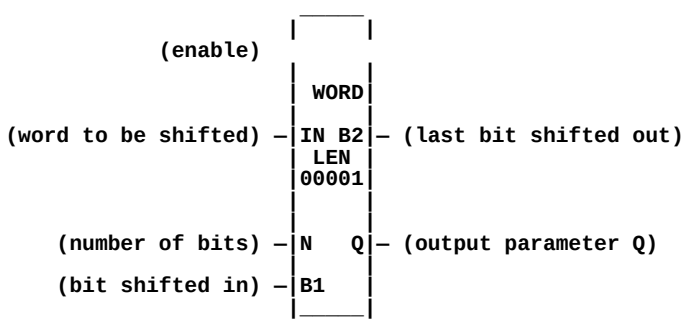
If the number of bits to be shifted (N) is greater than the number of bits in the array (LEN) * 16, or if the number of bits to be shifted is zero, then the array (Q) is filled with copies of the input bit (B1), and the input bit is copied to the output power flow (B2). If the number of bits to be shifted is zero, then no shifting is performed; the input array is copied into the output array; and input bit (B1) is copied into the power flow.

The bits being shifted into the beginning of the string are specified via input parameter B1. If a length greater than 1 has been specified as the number of bits to be shifted, each of the bits is filled with the same value (0 or 1). This can be:

- The boolean output of another program function.
- All 1s. To do this, use the special reference nickname ALW_ON as a permissive to input B1.
- All 0s. To do this, use the special reference nickname ALW_OFF as a permissive to input B1.

The SHL or SHR function passes power flow to the right, unless the number of bits specified to be shifted is zero.

Output Q is the shifted copy of the input string. If you want the input string to be shifted, the output parameter Q must use the same memory location as the input parameter IN. The entire shifted string is written on each scan that power is received. Output B2 is the last bit shifted out. For example, if four bits were shifted, B2 would be the fourth bit shifted out.



Parameters:

Parameter	Description
enable	When the function is enabled, the shift is performed.
IN	IN contains the first word to be shifted.
N	N contains the number of places (bits) that the array is to be shifted.
B1	B1 contains the bit value to be shifted into the array.
B2	B2 contains the bit value of the last bit shifted out of the array.
Q	Output Q contains the first word of the shifted array.
LEN	LEN is the number of words in the array to be shifted.

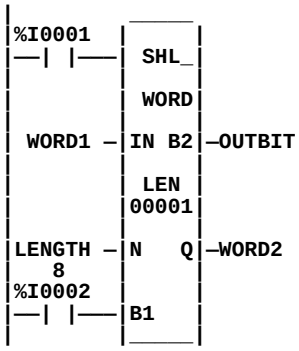
Valid Memory Types:

Parameter	flow	%I	%Q	%M	%T	%S	%G	%R	%AI	%AQ	const	none
enable	•											
IN		•	•	•	•	•	•	•	•	•		
N		•	•	•	•		•	•	•	•	•	
B1	•											
B2	•											•
Q		•	•	•	•	•†	•	•	•	•		

• Valid reference or place where power may flow through the function.
† %SA, %SB, or %SC only; %S cannot be used.

Example:

In the following example, whenever input %I0001 is set, the output bit string represented by the nickname WORD2 is made a copy of WORD1, left-shifted by the number of bits represented by the nickname LENGTH. The resulting open bits at the beginning of the output string are set to the value of %I0002.



ROL and ROR (WORD)

The Rotate Left (ROL) function is used to rotate all the bits in a string a specified number of places to the left. When rotation occurs, the specified number of bits is rotated out of the input string to the left and back into the string on the right.

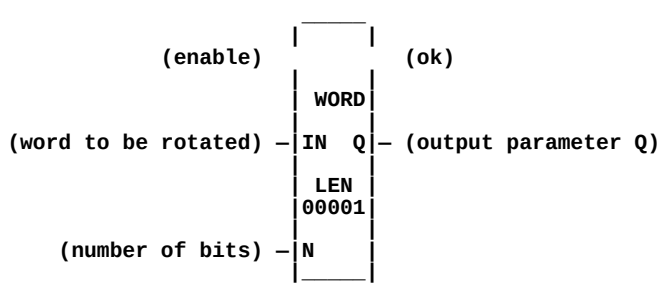
The Rotate Right (ROR) function rotates the bits in the string to the right. When rotation occurs, the specified number of bits is rotated out of the input string to the right and back into the string on the left.

A string length of 1 to 256 words can be selected for either function.

The number of places specified for rotation must be more than zero and less than the number of bits in the string. Otherwise, no movement occurs and no power flow is generated.

The ROL or ROR function passes power flow to the right, unless the number of bits specified to be rotated is greater than the total length of the string or is less than zero.

The result is placed in output string Q. If you want the input string to be rotated, the output parameter Q must use the same memory location as the input parameter IN. The entire rotated string is written on each scan that power is received.



Parameters:

Parameter	Description
enable	When the function is enabled, the rotation is performed.
IN	IN contains the first word to be rotated.
N	N contains the number of places that the array is to be rotated.
ok	The ok output is energized when the rotation is energized and the rotation length is not greater than the array size.
Q	Output Q contains the first word of the rotated array.
LEN	LEN is the number of words in the array to be rotated.

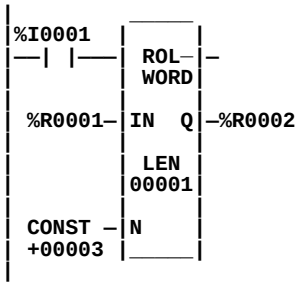
Valid Memory Types:

Parameter	flow	%I	%Q	%M	%T	%S	%G	%R	%AI	%AQ	const	none
enable	•											
IN		•	•	•	•	•	•	•	•	•		
N		•	•	•	•		•	•	•	•	•	
ok	•											•
Q		•	•	•	•	•†	•	•	•	•		

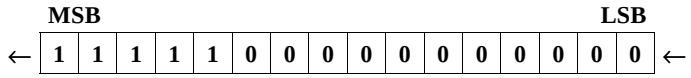
• Valid reference or place where power may flow through the function.
† %SA, %SB, or %SC only; %S cannot be used.

Example:

In the following example, whenever input %I0001 is set, the input bit string %R0001 is rotated 3 bits and the result is placed in %R0002. After execution of this function, the input bit string %R0001 is unchanged. If the same reference is used for IN and Q, a rotation will occur in place.



%R0001:



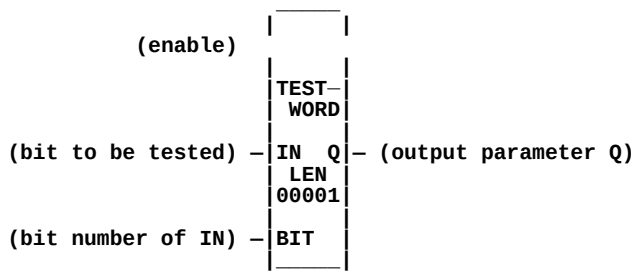
%R0002 (after %I0001 is set):

BTST (WORD)

The Bit Test (BTST) function is used to test a bit within a bit string to determine whether that bit is currently 1 or 0. The result of the test is placed in output Q.

Each sweep power is received, the BTST function sets its output Q to the same state as the specified bit. If a register rather than a constant is used to specify the bit number, the same function block can test different bits on successive sweeps. If the value of BIT is outside the range (1 ≤ BIT ≤ (16 * LEN)), then Q is set OFF.

A string length of 1 to 256 words can be selected.



Parameters:

Parameter	Description
enable	When the function is enabled, the bit test is performed.
IN	IN contains the first word of the data to be operated on.
BIT	BIT contains the bit number of IN that should be tested. Valid range is (1 ≤ BIT ≤ (16 * LEN)).
Q	Output Q is energized if the bit tested was a 1.
LEN	LEN is the number of words in the string to be tested.

Note

When using the Bit Test, Bit Set, Bit Clear or Bit Position function, the bits are numbered 1 through 16, ***NOT*** 0 through 15.

Valid Memory Types:

Parameter	flow	%I	%Q	%M	%T	%S	%G	%R	%AI	%AQ	const	none
enable	•											
IN		•	•	•	•	•	•	•	•	•		
BIT		•	•	•	•		•	•	•	•	•	
Q	•											•

• Valid reference or place where power may flow through the function.

Example:

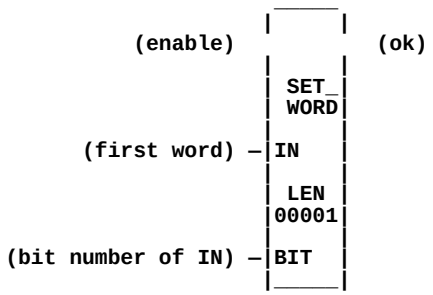
In the following example, whenever input %I0001 is set, the bit at the location contained in reference PICKBIT is tested. The bit is part of string PRD_CDE. If it is 1, output Q passes power flow, and the coil %Q0001 is turned on.

BSET and BCLR (WORD)

The Bit Set (BSET) function is used to set a bit in a bit string to 1. The Bit Clear (BCLR) function is used to clear a bit within a string by setting that bit to 0.

Each sweep that power is received, the function sets the specified bit to 1 for the BSET function or to 0 for the BCLR function. If a variable (register) rather than a constant is used to specify the bit number, the same function block can set different bits on successive sweeps.

A string length of 1 to 256 words can be selected. The function passes power flow to the right, unless the value for BIT is outside the range (1 ≤ BIT ≤ (16 * LEN)). Then, ok is set OFF.



Parameters:

Parameter	Description
enable	When the function is enabled, the bit operation is performed.
IN	IN contains the first word of the data to be operated on.
BIT	BIT contains the bit number of IN that should be set or cleared. Valid range is (1 ≤ BIT ≤ (16 * LEN)).
ok	The ok output is energized whenever enable is energized.
LEN	LEN is the number of words in the bit string.

Note

When using the Bit Test, Bit Set, Bit Clear or Bit Position function, the bits are numbered 1 through 16, ***NOT*** 0 through 15.

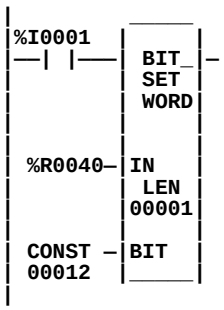
Valid Memory Types:

Parameter	flow	%I	%Q	%M	%T	%S	%G	%R	%AI	%AQ	const	none
enable	•											
IN		•	•	•	•	†	•	•	•	•		
BIT		•	•	•	•		•	•	•	•	•	
ok	•											•

• Valid reference or place where power may flow through the function.
† %SA, %SB, or %SC only; %S cannot be used.

Example:

In the following example, whenever input %I0001 is set, bit 12 of the string beginning at reference %R0040 is set to 1.



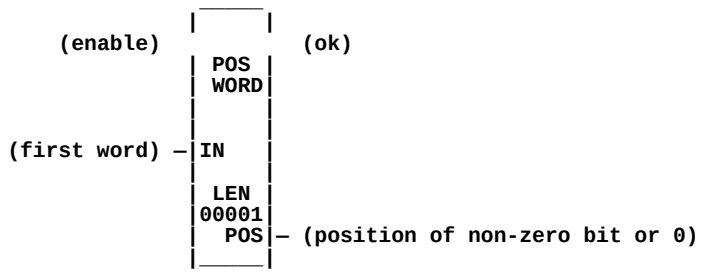
BPOS (WORD)

The Bit Position (BPOS) function is used to locate a bit set to 1 in a bit string.

Each sweep that power is received, the function scans the bit string starting at IN. When the function stops scanning, either a bit equal to 1 has been found or the entire length of the string has been scanned.

POS is set to the position within the bit string of the first non-zero bit; POS is set to zero if no non-zero bit is found.

A string length of 1 to 256 words can be selected. The function passes power flow to the right whenever enable is ON.



Parameters:

Parameter	Description
enable	When the function is enabled, a bit search operation is performed.
IN	IN contains the first word of the data to be operated on.
ok	The ok output is energized whenever enable is energized.
POS	The position of the first non-zero bit found, or zero if a non-zero bit is not found.
LEN	LEN is the number of words in the bit string.

Note

When using the Bit Test, Bit Set, Bit Clear or Bit Position function, the bits are numbered 1 through 16, ***NOT*** 0 through 15.

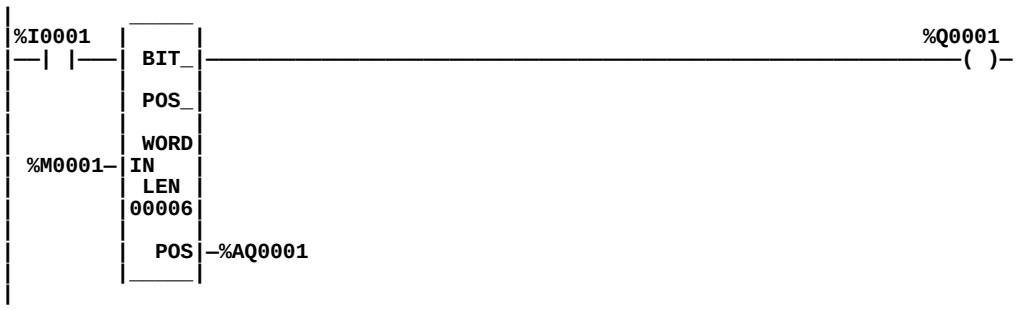
Valid Memory Types:

Parameter	flow	%I	%Q	%M	%T	%S	%G	%R	%AI	%AQ	const	none
enable	•											
IN		•	•	•	•	•	•	•	•	•		
POS		•	•	•	•		•	•	•	•		
ok	•											•

• Valid reference or place where power may flow through the function.

Example:

In the following example, if %I0001 is set, the bit string starting at %M0001 is searched until a bit equal to 1 is found, or 6 words have been searched. Coil %Q0001 is turned on. If a bit equal to 1 is found, its location within the bit string is written to %AQ001. If %I0001 is set, bit %M0001 is 0, and bit %M0002 is 1, then the value written to %AQ001 is 2.



MSKCMP (WORD, DWORD)

The Masked Compare (MSKCMP) function (*available for Release 4.41 or later CPUs*) is used to compare the contents of two separate bit strings with the ability to mask selected bits. The length of the bit strings to be compared is specified by the LEN parameter (where the value of LEN specifies the number of 16-bit words for the MSKCMPW function and 32-bit words for the MSKCMPD function).

When the logic controlling the enable input to the function passes power flow to the enable (EN) input, the function begins comparing the bits in the first string with the corresponding bits in the second string. Comparison continues until a miscompare is found, or until the end of the string is reached.

The BIT input is used to store the bit number where the next comparison should start (where a *0* indicates the first bit in the string). The BN output is used to store the bit number where the last comparison occurred (where a *1* indicates the first bit in the string). Using the same reference for BIT and BN causes the compare to start at the next bit position after a miscompare; or, if all bits compared successfully upon the next invocation of the function block, the compare starts at the beginning.

If you want to start the next comparison at some other location in the string, you can enter different references for BIT and BN. If the value of BIT is a location that is beyond the end of the string, BIT is reset to 0 before starting the next comparison.

If All Bits in I1 and I2 are the Same

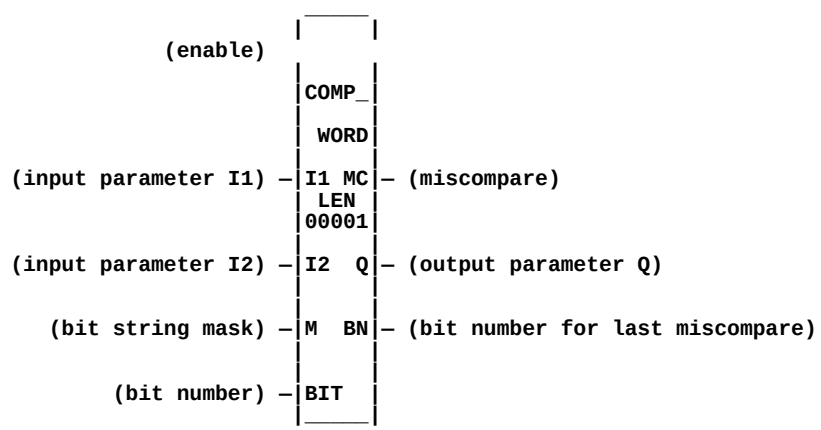
If all corresponding bits in strings I1 and I2 match, the function sets the “miscompare” output MC to 0 and BN to the highest bit number in the input strings. The comparison then stops. On the next invocation of MSKCMPW, it will be reset to 0.

If a Miscompare is Found

When the two bits currently being compared are not the same, the function checks the correspondingly numbered bit in string M (the mask). If the mask bit is a *1*, the comparison continues until it reaches another miscompare or the end of the input strings.

If a miscompare is detected and the corresponding mask bit is a 0, the function does the following:

1. Sets the corresponding mask bit in M to 1.
2. Sets the miscompare (MC) output to 1.
3. Updates the output bit string Q to match the new content of mask string M.
4. Sets the bit number output (BN) to the number of the miscompared bit.
5. Stops the comparison.



Parameters:

Parameter	Description
enable	Permissive logic to enable the function.
I1	Reference for the first bit string to be compared.
I2	Reference for the second bit string to be compared.
M	Reference for the bit string mask.
BIT	Reference for the bit number where the next comparison should start.
MC	User logic to determine if a miscompare has occurred.
Q	Output copy of the mask (M) bit string.
BN	Number of the bit where the last compare occurred.
LEN	LEN is the number of words in the bit string.

Valid Memory Types:

Parameter	flow	%I	%Q	%M	%T	%S	%G	%R	%AI	%AQ	const	none
enable	•											
I1		o	o	o	o	o	o	•	•	•		
I2		o	o	o	o	o	o	•	•	•		
M		o	o	o	o	o†	o	•	•	•		
BIT		•	•	•	•	•	•	•	•	•	•	
LEN											•‡	
MC	•											•
Q		o	o	o	o	o†	o	•	•	•		
BN		•	•	•	•	•	•	•	•	•		

• Valid reference or place where power may flow through the function.
o Valid reference for WORD data only; not valid for DWORD.
† %SA, %SB, %SC only; %S cannot be used.
‡ Max const value of 4095 for WORD and 2047 for DWORD.

Example:

In the following example, after first scan, the MSKCMPW function block is executed. %M0001 through %M0016 is compared with %M0017 through %M0032. %M0033 through %M0048 contains the mask value. The value in %R0001 determines at which bit position the comparison starts within the two input strings. The contents of the above references before the function block is executed are as follows:

(I1) – %M0001 = 6C6Ch =

0	1	1	0	1	1	0	0	0	1	1	0	1	1	0	0

(I2) – %M0017 = 606Fh =

0	1	1	0	1	1	0	1	0	1	1	0	1	1	1	1

(M/Q) – %M0033 = 000Fh =

0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1

(BIT/BN) – %R0001 = 0
(MC) – %Q0001 = OFF

The contents of these references after the function block is executed are as follows:

(I1) – %M0001 =

0	1	1	0	1	1	0	0	0	1	1	0	1	1	0	0

(I2) – %M0017 =

0	1	1	0	1	1	0	1	0	1	1	0	1	1	1	1

(M/Q) – %M0033

0	0	0	0	0	0	0	1	0	0	0	0	1	1	1	1

(BIT/BN) – %R0001 = 8

(MC) – %Q0001 = ON

Ladder Diagram Representation

Notice that, in the example shown above, we used the FST_SCN contact to force one and only one execution; otherwise the masked compare would repeat, not necessarily delivering the desired results.

Section 6: Data Move Functions

Data move functions provide basic data move capabilities. This section describes the following data move functions:

Abbreviation	Function	Description	Page
MOVE	Move	Copy data as individual bits. The maximum length allowed is 256 words, except MOVE_BIT is 256 bits. Data can be moved into a different data type without prior conversion.	4-70
BLKMOV	Block Move	Copy a block of seven constants to a specified memory location. The constants are input as part of the function.	4-73
BLKCLR	Block Clear	Replace the content of a block of data with all zeros. This function can be used to clear an area of bit (%I, %Q, %M, %G, or %T) or word (%R, %AI, or %AQ) memory. The maximum length allowed is 256 words.	4-75
SHFR	Shift Register	Shift one or more data words into a table. The maximum length allowed is 256 words.	4-77
BITSEQ	Bit Sequencer	Perform a bit sequence shift through an array of bits. The maximum length allowed is 256 words.	4-80
COMMREQ	Communications Request	Allow the program to communicate with an intelligent module, such as a Genius Communications Module or a Programmable Coprocessor Module.	4-83

MOVE (BIT, INT, WORD, REAL)

Use the MOVE function to copy data (as individual bits) from one location to another. Because the data is copied in bit format, the new location does not need to be the same data type as the original location.

The MOVE function has two input parameters and two output parameters. When the function receives power flow, it copies data from input parameter IN to output parameter Q as bits. If data is moved from one location in discrete memory to another, (for example, from %I memory to %T memory), the transition information associated with the discrete memory elements is updated to indicate whether or not the MOVE operation caused any discrete memory elements to change state. Data at the input parameter does not change unless there is an overlap in the source and destination.

For the BIT type there is another consideration. If a BIT array specified on the Q parameter does not encompass all of the bits in a byte, the transition bits associated with that byte (which are not in the array) will be cleared when the MOVE_BIT receives power flow.

Input IN can be either a reference for the data to be moved or a constant. If a constant is specified, then the constant value is placed in the location specified by the output reference. For example, if a constant value of 4 is specified for IN, then 4 is placed in the memory location specified by Q. If the length is greater than 1 and a constant is specified, then the constant is placed in the memory location specified by Q and the locations following, up to the length specified. For example, if the constant value 9 is specified for IN and the length is 4, then 9 is placed in the memory location specified by Q and the three locations following.

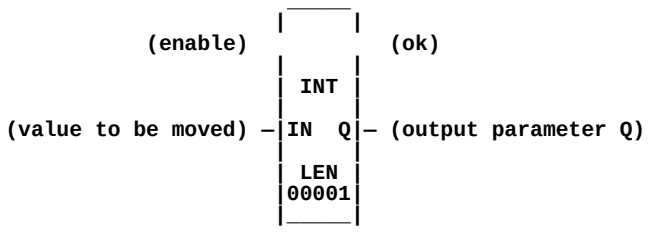
The LEN operand specifies the number of:

- Words to be moved for MOVE_INT and MOVE_WORD.
- Bits to be moved for MOVE_BIT.
- Reals to be moved for MOVE_REAL.

Note

The REAL data type is only available on 350 and 360 series CPUs, Release 9 or later, or on all releases of CPU352.

The function passes power to the right whenever power is received.



Parameters:

Parameter	Description
enable	When the function is enabled, the move is performed.
IN	IN contains the value to be moved. For MOVE_BIT, any discrete reference may be used; it does not need to be byte aligned. However, 16 bits, beginning with the reference address specified, are displayed online.
ok	The ok output is energized whenever the function is enabled.
Q	When the move is performed, the value at IN is written to Q. For MOVE_BIT, any discrete reference may be used; it does not need to be byte aligned. However, 16 bits, beginning with the reference address specified, are displayed online.
LEN	LEN specifies the number of words or bits to be moved. For MOVE_WORD and MOVE_INT, LEN must be between 1 and 256 words. For MOVE_BIT, when IN is a constant, LEN must be between 1 and 16 bits; otherwise, LEN must be between 1 and 256.

Note

On 351, 352 and 360 series CPUs, the MOVE_INT and MOVE_WORD functions do not support overlapping of IN and Q parameters, where the IN reference is less than the Q reference. For example, with the following values: IN=%R0001, Q=%R0004, LEN=5 (words), the %R0007 and %R0008 contents will be indeterminate; however, using the following values: Q=%R0001, IN=%R0004, LEN=5 (words) will yield valid contents.

Also, please note that only 350 and 360 series CPUs (Release 9 and later, plus all releases of CPU352) have Floating Point capabilities at this time and therefore the only one capable of MOVE_REAL.

Valid Memory Types:

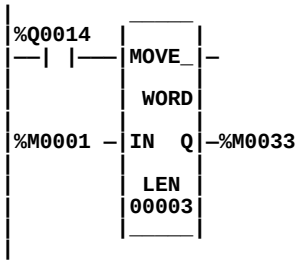
Parameter	flow	%I	%Q	%M	%T	%S	%G	%R	%AI	%AQ	const	none
enable	•											
IN		•	•	•	•	o	•	•	•	•	•	
ok	•											•
Q		•	•	•	•	o†	•	•	•	•		

Note: For REAL data, the only valid types are %R, %AI, and %AQ.

- • Valid reference for BIT, INT, or WORD data, or place where power may flow through the function. For MOVE_BIT, discrete user references %I, %Q, %M, and %T need not be byte aligned.
- o Valid reference for BIT or WORD data only; not valid for INT.
- † %SA, %SB, %SC only; %S cannot be used.

Example 1:

When enabling input %Q0014 is ON, 48 bits are moved from memory location %M0001 to memory location %M0033. Even though the destination overlaps the source for 16 bits, the move is done correctly (except for the 351 and 352 CPUs as noted on previously).



Before using the Move function:

INPUT (%M0001 through %M0048)

															1
%M0016	1	1	1	1	0	0	0	0	1	1	1	1	0	0	0	0
%M0032	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
%M0048	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

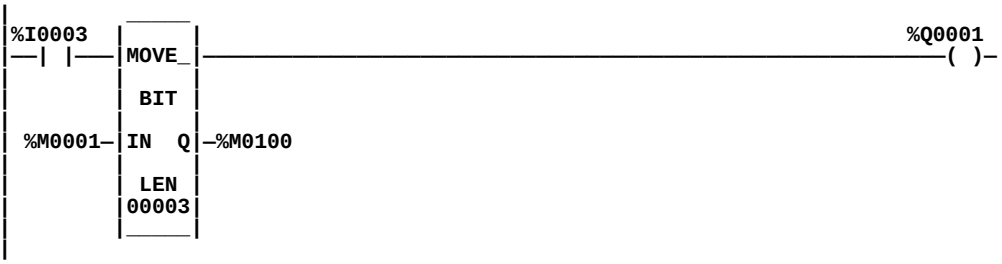
After using the Move function:

INPUT (%M0033 through %M0080)

															33
%M0048	1	1	1	1	0	0	0	0	1	1	1	1	0	0	0	0
%M0064	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
%M0080	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

Example 2:

In this example, whenever %I0001 is set, the three bits %M0001, %M0002, and %M0003 are moved to %M0100, %M0101, and %M0102, respectively. Coil %Q0001 is turned on.



BLKMOV (INT, WORD, REAL)

Use the Block Move (BLKMOV) function to copy a block of seven constants to a specified location.

Note

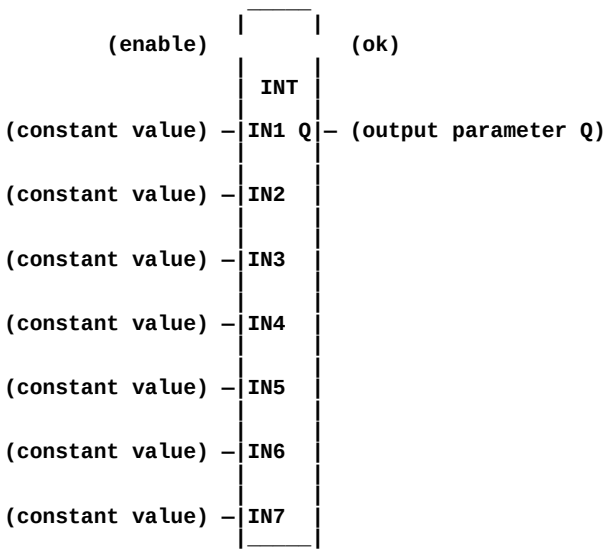
The REAL data type is only available on 350 and 360 series CPUs, Release 9 or later, or on all releases of CPU352.

The BLKMOV function has eight input parameters and two output parameters. When the function receives power flow, it copies the constant values into consecutive locations, beginning at the destination specified in output Q. Output Q cannot be the input of another program function.

Note

For BLKMOV_INT, the values of IN1 — IN7 are displayed as signed decimals. For BLKMOV_WORD, IN1 — IN7 are displayed in hexadecimal. For BLKMOV_REAL, IN1— IN7 are displayed in Real format.

The function passes power to the right whenever power is received.



Parameters:

Parameter	Description
enable	When the function is enabled, the block move is performed.
IN1— IN7	IN1 through IN7 contain seven constant values.
ok	The ok output is energized whenever the function is enabled.
Q	Output Q contains the first integer of the moved array. IN1 is moved to Q.

Valid Memory Types:

Parameter	flow	%I	%Q	%M	%T	%S	%G	%R	%AI	%AQ	const	none
enable	•											
IN1 — IN7											•	
ok	•											•
Q		•	•	•	•	o†	•	•	•	•		

Note: For REAL data, the only valid types are %R, %AI, and %AQ.

• Valid reference for place where power may flow through the function.

o Valid reference for WORD data only; not valid for INT or REAL.

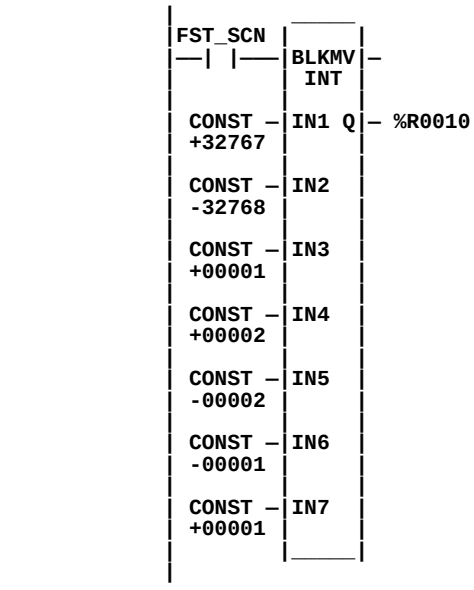
† %SA, %SB, %SC only; %S cannot be used.

Note

Floating Point capabilities exist only on 350 and 360 series CPUs, Release 9 or later, or on all releases of CPU352. These 90-30 CPUs are the only ones capable of BLKMV_REAL.

Example:

In the following example, when the enabling input represented by the nickname FST_SCN is ON, the BLKMOV function copies the seven input constants into memory locations %R0010 through %R0016.

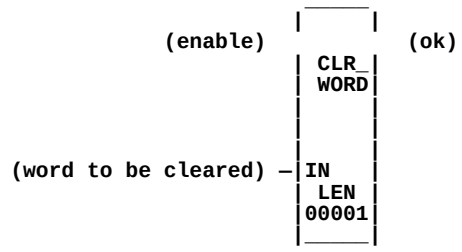


BLKCLR (WORD)

Use the Block Clear (BLKCLR) function to fill a specified block of data with zeros.

The BLKCLR function has two input parameters and one output parameter. When the function receives power flow, it writes zeros into the memory location beginning at the reference specified by IN. When the data to be cleared is from discrete memory (%I, %Q, %M, %G, or %T), the transition information associated with the references is also cleared.

The function passes power to the right whenever power is received.



Parameters:

Parameter	Description
enable	When the function is enabled, the array is cleared.
IN	IN contains the first word of the array to be cleared.
ok	The ok output is energized whenever the function is enabled.
LEN	LEN must be between 1 and 256 words.

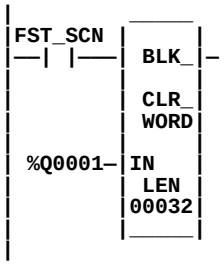
Valid Memory Types:

Parameter	flow	%I	%Q	%M	%T	%S	%G	%R	%AI	%AQ	const	none
enable	•											
IN		•	•	•	•	•†	•	•	•	•		
ok	•											•

• Valid reference or place where power may flow through the function.
† %SA, %SB, %SC only; %S cannot be used.

Example:

In the following example, at power-up, 32 words of %Q memory (512 points) beginning at %Q0001 are filled with zeros.



SHFR (BIT, WORD)

Use the Shift Register (SHFR) function to shift one or more data words or data bits from a reference location into a specified area of memory. For example, one word might be shifted into an area of memory with a specified length of five words. As a result of this shift, another word of data would be shifted out of the end of the memory area.

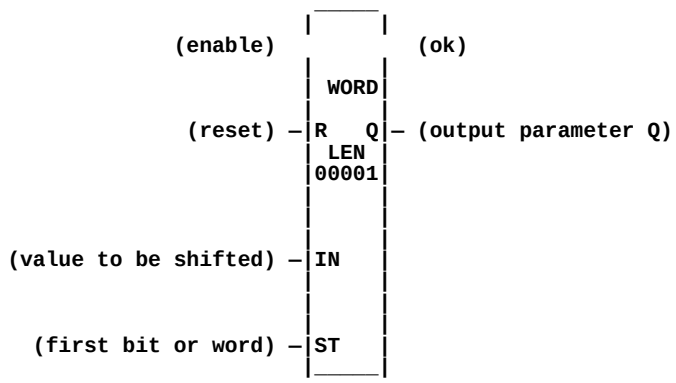
Note

When assigning reference addresses, overlapping input and output reference address ranges in multi-word functions may produce unexpected results.

The SHFR function has four input parameters and two output parameters. The reset input (R) takes precedence over the function enable input. When the reset is active, all references beginning at the shift register (ST) up to the length specified for LEN, are filled with zeros.

If the function receives power flow and reset is not active, each bit or word of the shift register is moved to the next highest reference. The last element in the shift register is shifted into Q. The highest reference of the shift register element of IN is shifted into the vacated element starting at ST. The contents of the shift register are accessible throughout the program because they are overlaid on absolute locations in logic addressable memory.

The function passes power to the right whenever power is received through the enable logic.



Parameters:

Parameter	Description
enable	When enable is energized and R is not, the shift is performed.
R	When R is energized, the shift register located at ST is filled with zeros.
IN	IN contains the value to be shifted into the first bit or word of the shift register. For SHFR_BIT, any discrete reference may be used; it does not need to be byte aligned. However, 16 bits, beginning with the reference address specified, are displayed online.
ST	ST contains the first bit or word of the shift register. For SHFR_BIT, any discrete reference may be used; it does not need to be byte aligned. However, 16 bits, beginning with the reference address specified, are displayed online.
ok	The ok output is energized whenever the function is enabled and R is not enabled.
Q	Output Q contains the bit or word shifted out of the shift register. For SHFR_BIT, any discrete reference may be used; it does not need to be byte aligned. However, 16 bits, beginning with the reference address specified, are displayed online.
LEN	LEN determines the length of the shift register. For SHFR_WORD, LEN must be between 1 and 256 words. For SHFR_BIT, LEN must be between 1 and 256 bits.

Valid Memory Types:

Parameter	flow	%I	%Q	%M	%T	%S	%G	%R	%AI	%AQ	const	none
enable	•											
R	•											
IN		•	•	•	•	•	•	•	•	•	•	
ST		•	•	•	•	•†	•	•	•	•		
ok	•											•
Q		•	•	•	•	•†	•	•	•	•		

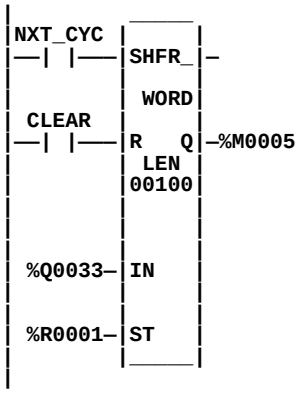
• Valid reference for BIT or WORD data, or place where power may flow through the function.
For SHFR_BIT, discrete user references %I, %Q, %M, and %T need not be byte aligned.

† %SA, %SB, %SC only; %S cannot be used.

Example 1:

In the following example, the shift register operates on register memory locations %R0001 through %R0100. When the reset reference CLEAR is active, the shift register words are set to zero.

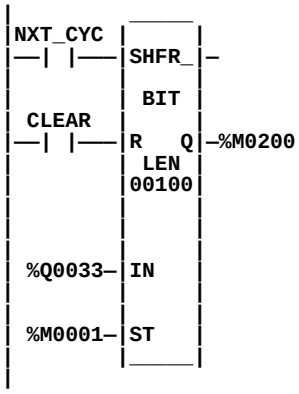
When the NXT_CYC reference is active and CLEAR is not active, the word from output status table location %Q0033 is shifted into the shift register at %R0001. The word shifted out of the shift register from %R0100 is stored in output %M0005.



Example 2:

In this example, the shift register operates on memory locations %M0001 through %M0100. When the reset reference CLEAR is active, the SHFR function fills %M0001 through %M0100 with zeros.

When NXT_CYC is active and CLEAR is not, the SHFR function shifts the data in %M0001 to %M0100 down by one bit. The bit in %Q0033 is shifted into %M0001 while the bit shifted out of %M0100 is written to %M0200.



BITSEQ (BIT)

The Bit Sequencer (BITSEQ) function performs a bit sequence shift through an array of bits. The BITSEQ function has five input parameters and one output parameter. The operation of the function depends on the previous value of the parameter EN, as shown in the following table.

R Current Execution	EN Previous Execution	EN Current Execution	Bit Sequencer Execution
OFF	OFF	OFF	Bit sequencer does not execute.
OFF	OFF	ON	Bit sequencer increments/decrements by 1.
OFF	ON	OFF	Bit sequencer does not execute.
OFF	ON	ON	Bit sequencer does not execute.
ON	ON/OFF	ON/OFF	Bit sequencer resets.

The reset input (R) overrides the enable (EN) and always resets the sequencer. When R is active, the current step number is set to the value passed in via the step number parameter. If no step number is passed in, step is set to 1. All of the bits in the sequencer are set to 0, except for the bit pointed to by the current step, which is set to 1.

When EN is active and R is not active, the bit pointed to by the current step number is cleared. The current step number is either incremented or decremented, based on the direction parameter. Then, the bit pointed to by the new step number is set to 1.

- When the step number is being incremented and it goes outside the range of (1 ≤ step number ≤ LEN), it is set back to 1.
- When the step number is being decremented and it goes outside the range of (1 ≤ step number ≤ LEN), it is set to LEN.

The parameter ST is optional. If it is not used, the BITSEQ operates as described above, except that no bits are set or cleared. Basically, the BITSEQ then just cycles the current step number through its legal range.

Memory Required for a Bit Sequencer

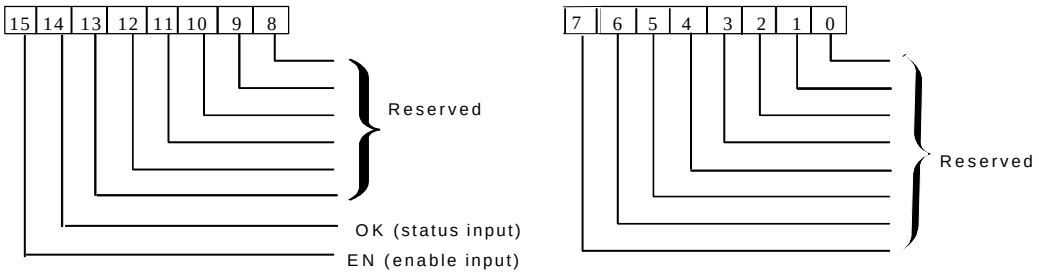
Each bit sequencer uses three words (registers) of %R memory to store the following information:

current step number	word 1
length of sequence (in bits)	word 2
control word	word 3

When you enter a bit sequencer, you must enter a beginning address for these three words (registers) directly below the graphic representing the function (see example on next page).

```
                                  ┌─────┐
                                  │     │
(enable)                          │     │  (ok)
                                  │ SEQ │
                                  │     │
(reset) ──                        │R    │
                                  │ LEN │
                                  │00001│
(direction) ──                    │DIR  │
                                  │     │
                                  │     │
(number) ──                       │STEP │
                                  │     │
                                  │     │
(starting address) ──             │ST   │
                                  │     │
                                  │     │
                                  └─────┘
                                 (address)  - Enter the beginning address here.
```

The control word stores the state of the boolean inputs and outputs of its associated function block, as shown in the following format:



Note

Bits 0 through 13 are not used. Also, note that bits need to be entered as 1 through 16, ***NOT*** 0 through 15 in the STEP parameter.

Parameters:

Parameter	Description
address	Address is the location of the bit sequencer's current step, length, and the last enable and ok statuses.
enable	When the function is enabled, if it was not enabled on the previous sweep and if R is not energized, the bit sequence shift is performed.
R	When R is energized, the bit sequencer's step number is set to the value in STEP (default = 1), and the bit sequencer is filled with zeros, except for the current step number bit.
DIR	When DIR is energized, the bit sequencer's step number is incremented prior to the shift. Otherwise, it is decremented.
STEP	When R is energized, the step number is set to this value.
ST	ST contains the first word of the bit sequencer.
ok	The ok output is energized whenever the function is enabled.
LEN	LEN must be between 1 and 256 bits.

Note

Coil checking, for the BITSEQ function, checks for 16 bits from the ST parameter, even when LEN is less than 16.

Valid Memory Types:

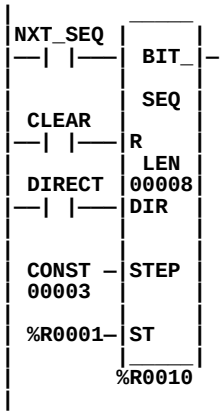
Parameter	flow	%I	%Q	%M	%T	%S	%G	%R	%AI	%AQ	const	none
address								•				
enable	•											
R	•											
DIR	•											
STEP		•	•	•	•		•	•	•	•	•	•
ST		•	•	•	•	•†	•	•	•	•		•
ok	•											•

• Valid reference or place where power may flow through the function.
† SA, %SB, %SC only; %S cannot be used

Example:

In the following example, the sequencer operates on register memory %R0001. Its static data is stored in registers %R0010, %R0011, and %R0012. When CLEAR is active, the sequencer is reset and the current step is set to step number 3. The first 8 bits of %R0001 are set to zero.

When NXT_SEQ is active and CLEAR is not active, the bit for step number 3 is cleared and the bit for step number 2 or 4 (depending on whether DIR is energized) is set.



COMMREQ

Use the Communication Request (COMMREQ) function if the program needs to communicate with an intelligent module, such as a Genius Communications Module or a Programmable Coprocessor Module.

Note

The information presented on the following pages shows the format of the COMMREQ function. You will need additional information to program the COMMREQ for each type of device. Programming requirements for each module that uses the COMMREQ function are described in the module's documentation.

The COMMREQ function has three input parameters and one output parameter. When the COMMREQ function receives power flow, a command block of data is sent to the intelligent module. The command block begins at the reference specified using the parameter IN. The rack and slot # of the intelligent module is specified in SYSID.

The COMMREQ may either send a message and wait for a reply, or send a message and continue without waiting for a reply. If the command block specifies that the program will not wait for a reply, the command block contents are sent to the receiving device and the program execution resumes immediately. (The timeout value is ignored.) This is referred to as **NOWAIT** mode.

If the command block specifies that the program will wait for a reply, the command block contents are sent to the receiving device and the CPU waits for a reply. The maximum length of time the PLC will wait for the device to respond is specified in the command block. If the device does not respond within that time, program execution resumes. This is referred to as **WAIT** mode.

The Function Faulted (FT) output may be set ON if:

1. The specified target address is not present (SYSID).
2. The specified task is not valid for the device (TASK).
3. The data length is 0.
4. The device's status pointer address (part of the command block) does not exist. This may be due to an incorrect memory type selection, or an address within that memory type that is out of range.

Command Block

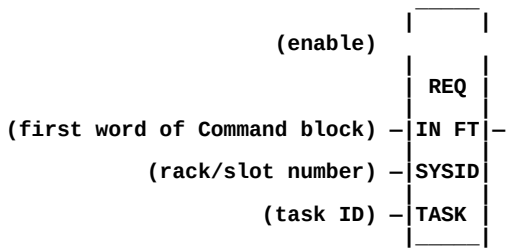
The command block provides information to the intelligent module on the command to be performed.

The address of the command block is specified for the IN input to the COMMREQ function. This address may be in any word-oriented area of memory (%R, %AI, or %AQ). The length of the command block depends on the amount of data sent to the device.

The command block has the following structure:

Length (in words)	address
Wait/No Wait Flag	address + 1
Status Pointer Memory	address + 2
Status Pointer Offset	address + 3
Idle Timeout Value	address + 4
Maximum Communication Time	address + 5
Data Block	address + 6 to address + 133

Information required for the command block can be placed in the designated memory area using an appropriate programming function.



Parameters:

Parameter	Description
enable	When the function is energized, the communications request is performed.
IN	IN contains the first word of the command block.
SYSID	SYSID contains the rack number (most significant byte) and slot number (least significant byte) of the target device.
TASK	TASK contains the task ID of the process on the target device.
FT	FT is energized if an error is detected processing the COMMREQ.

Valid Memory Types:

Parameter	flow	%I	%Q	%M	%T	%S	%G	%R	%AI	%AQ	const	none
enable	•											
IN								•	•	•		
SYSID		•	•	•	•		•	•	•	•	•	
TASK								•	•	•	•	
FT	•											•

• Valid reference or place where power may flow through the function.

Example:

In the following example, when enabling input %M0020 is ON, a command block located starting at %R0016 is sent to communications task 1 in the device located at rack 1, slot 2 of the PLC. If an error occurs processing the COMMREQ, %Q0100 is set.

Note

For systems that do not have expansion racks, the SYSID must be zero for the main rack.

Section 7: Table Functions

Table functions are used to perform the following functions:

Abbreviation	Function	Description	Page
ARRAY_MOVE	Array Move	Copy a specified number of data elements from a source array to a destination array.	4-87
SRCH_EQ	Search Equal	Search for all array values equal to a specified value.	4-91
SRCH_NE	Search Not Equal	Search for all array values not equal to a specified value.	4-91
SRCH_GT	Search Greater Than	Search for all array values greater than a specified value.	4-91
SRCH_GE	Search Greater Than or Equal	Search for all array values greater than or equal to a specified value.	4-91
SRCH_LT	Search Less Than	Search for all array values less than a specified value.	4-91
SRCH_LE	Search Less Than or Equal	Search for all array values less than or equal to a specified value.	4-91

The maximum length allowed for these functions is 32,767 bytes or words, or 262,136 bits (bits are available for ARRAY_MOVE only).

Table functions operate on these types of data:

Data Type	Description
INT	Signed integer.
DINT	Double precision signed integer.
BIT *	Bit data type.
BYTE	Byte data type.
WORD	Word data type.

* Only available for ARRAY_MOVE.

The default data type is signed integer. The data type can be changed after selecting the specific data table function. To compare data of other types or of two different types, first use the appropriate conversion function (described in section 8, “Conversion Functions”) to change the data to one of the data types listed above.

ARRAY_MOVE (INT, DINT, BIT, BYTE, WORD)

Use the Array Move (ARRAY_MOVE) function to copy a specified number of data elements from a source array to a destination array.

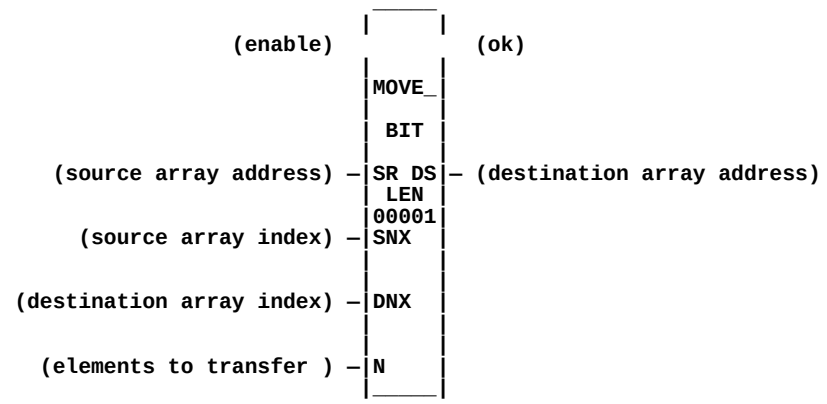
The ARRAY_MOVE function has five input parameters and two output parameters. When the function receives power flow, the number of data elements in the count indicator (N) is extracted from the input array starting with the indexed location (SR + SNX — 1). The data elements are written to the output array starting with the indexed location (DS + DNX — 1). The LEN operand specifies the number of elements that make up each array.

For ARRAY_MOVE_BIT, when word-oriented memory is selected for the parameters of the source array and/or destination array starting address, the least significant bit of the specified word is the first bit of the array. The value displayed contains 16 bits, regardless of the length of the array.

The indices in an ARRAY_MOVE instruction are 1-based. In using an ARRAY_MOVE, no element outside either the source or destination arrays (as specified by their starting address and length) may be referenced.

The ok output will receive power flow, unless one of the following conditions occurs:

- Enable is OFF.
- (N + SNX – 1) is greater than LEN.
- (N + DNX – 1) is greater than LEN.



Parameters:

Parameter	Description
enable	When the function is enabled, the operation is performed.
SR	SR contains the starting address of the source array. For ARRAY_MOVE_ BIT, any reference may be used; it does not need to be byte aligned. However, 16 bits, beginning with the reference address specified, are displayed online.
SNX	SNX contains the index of the source array.
DNX	DNX contains the index of the destination array.
N	N provides a count indicator.
ok	The ok output is energized whenever enable is energized.
DS	DS contains the starting address of the destination array. For ARRAY_MOVE_ BIT, any reference may be used; it does not need to be byte aligned. However, 16 bits, beginning with the reference address specified, are displayed online.
LEN	LEN specifies the number of elements starting at SR and DS that make up each array.

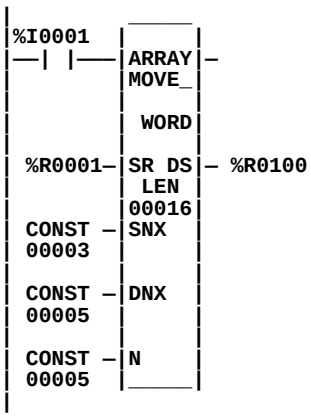
Valid Memory Types:

Parameter	flow	%I	%Q	%M	%T	%S	%G	%R	%AI	%AQ	const	none
enable	•											
SR		o	o	o	o	Δ†	o	•	•	•		
SNX		•	•	•	•		•	•	•	•	•	
DNX		•	•	•	•		•	•	•	•	•	
N		•	•	•	•		•	•	•	•	•	
ok	•											•
DS		o	o	o	o	†	o	•	•	•		

• Valid reference or place where power may flow through the function.
For ARRAY_MOVE_BIT, discrete user references %I, %Q, %M, and %T need not be byte aligned.
o Valid reference for INT, BIT, BYTE, or WORD data only; not valid for DINT.
Δ Valid data type for BIT, BYTE, or WORD data only; not valid for INT or DINT.
† %SA, %SB, %SC only; %S cannot be used.

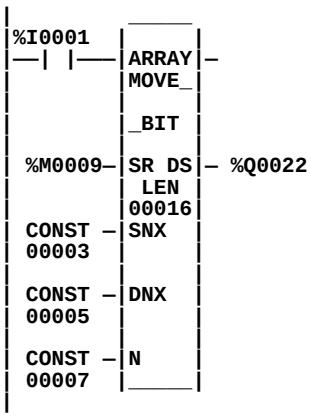
Example 1:

In this example, %R0003 — %R0007 of the array %R0001 — %R0016 is read and then written into %R0104 — %R0108 of the array %R0100 — %R0115.



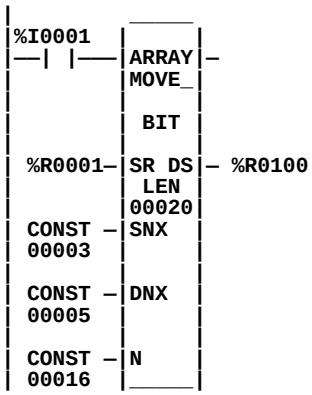
Example 2:

Using bit memory for SR and DS, %M0011— %M0017 of the array %M009 — %M0024 is read and then written to %Q0026 — %Q0032 of the array %Q0022 — %Q0037.



Example 3:

Using word memory, for SR and DS, the third least significant bit of %R0001 through the second least significant bit of %R0002 of the array containing all 16 bits of %R0001 and four bits of %R0002 is read and then written into the fifth least significant bit of %R0100 through the fourth least significant bit of %R0101 of the array containing all 16 bits of %R0100 and four bits of %R0101.



SRCH_EQ and SRCH_NE (INT, DINT, BYTE, WORD)
SRCH_GT and SRCH_LT
SRCH_GE and SRCH_LE

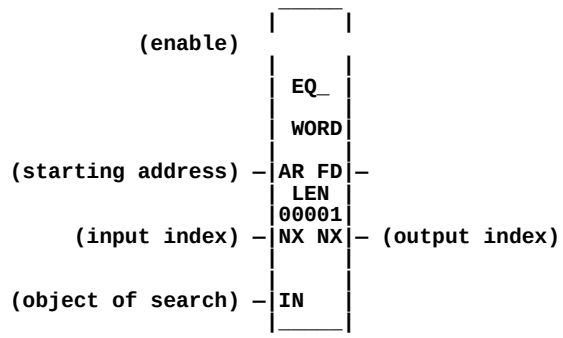
Use the appropriate Search function listed below to search for all array values for that particular operation.

Abbreviation	Function	Description
SRCH_EQ	Search Equal	Search for all array values equal to a specified value.
SRCH_NE	Search Not Equal	Search for all array values not equal to a specified value.
SRCH_GT	Search Greater Than	Search for all array values greater than a specified value.
SRCH_GE	Search Greater Than or Equal	Search for all array values greater than or equal to a specified value.
SRCH_LT	Search Less Than	Search for all array values less than a specified value.
SRCH_LE	Search Less Than or Equal	Search for all array values less than or equal to a specified value.

Each function has four input parameters and two output parameters. When the function receives power, the array is searched starting at (AR + input NX). This is the starting address of the array (AR) plus the index into this array (input NX).

The search continues until the array element of the search object (IN) is found or until the end of the array is reached. If an array element is found, output parameter (FD) is set ON and output parameter (output NX) is set to the relative position of this element within the array. If no array element is found before the end of the array is reached, then output parameter (FD) is set OFF and output parameter (output NX) is set to zero.

The valid values for input NX are 0 to LEN — 1. NX should be set to zero to begin searching at the first element. This value increments by one at the time of execution. Therefore, the values of output NX are 1 to LEN. If the value of input NX is out-of-range, (< 0 or ≥ LEN), its value is set to the default value of zero.



Parameters:

Parameter	Description
enable	When the function is enabled, the operation is performed.
AR	AR contains the starting address of the array to be searched.
Input NX	Input NX contains the index into the array at which to begin the search.
IN	IN contains the object of the search.
Output NX	Output NX holds the position within the array of the search target.
FD	FD indicates that an array element has been found and the function was successful.
LEN	LEN specifies the number of elements starting at AR that make up the array to be searched. It may be 1 to 32,767 bytes or words.

Valid Memory Types:

Parameter	flow	%I	%Q	%M	%T	%S	%G	%R	%AI	%AQ	const	none
enable	•											
AR		o	o	o	o	Δ	o	•	•	•		
NX in		•	•	•	•		•	•	•	•	•	
IN		o	o	o	o	Δ	o	•	•	•	•	
NX out		•	•	•	•		•	•	•	•		
FD	•											•

• Valid reference or place where power may flow through the function.
o Valid reference for INT, BYTE, or WORD data only; not valid for DINT.
Δ Valid reference for BYTE or WORD data only; not valid for INT or DINT.

Example 1:

The array AR is defined as memory addresses %R0001 — %R0005. When EN is ON, the portion of the array between %R0004 and %R0005 is searched for an element whose value is equal to IN. If %R0001 = 7, %R0002 = 9, %R0003 = 6, %R0004 = 7, %R0005 = 7, and %R0100 = 7, then the search will begin at %R0004 and conclude at %R0004 when FD is set ON and a 4 is written to %R0101.

Example 2:

Array AR is defined as memory addresses %AI0001 — %AI0016. The values of the array elements are 100, 20, 0, 5, 90, 200, 0, 79, 102, 80, 24, 34, 987, 8, 0, and 500. Initially, %AQ0001 is 5. When EN is ON, each sweep will search the array looking for a match to the IN value of 0. The first sweep will start searching at %AI0006 and find a match at %AI0007, so FD is ON and %AQ0001 is 7. The second sweep will start searching at %AI0008 and find a match at %AI0015, so FD remains ON and %AQ0001 is 15. The next sweep will start at %AI0016. Since the end of the array is reached without a match, FD is set OFF and %AQ0001 is set to zero. The next sweep will start searching at the beginning of the array.

Section 8: Conversion Functions

Use the conversion functions to convert a data item from one number type to another. Many programming instructions, such as math functions, must be used with data of one type. This section describes the following conversion functions:

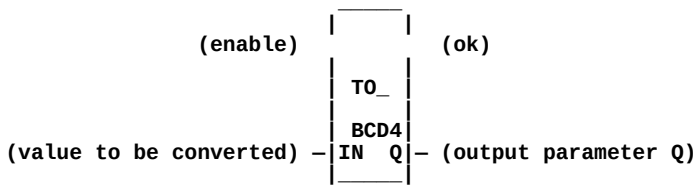
Abbreviation	Function	Description	Page
BCD-4	Convert to BCD-4	Convert a signed integer to 4-digit BCD format.	4-95
INT	Convert to Signed Integer	Convert BCD-4 or REAL to signed integer.	4-97
DINT	Convert to Double Precision Signed Integer	Convert REAL to double precision signed integer format.	4-99
REAL	Convert to REAL	Convert INT, DINT, BCD-4, or WORD to REAL.	4-101
WORD	Convert to WORD	Convert REAL to WORD format.	4-103
TRUN	Truncate	Round the real number toward zero.	4-105

—>BCD-4 (INT)

The Convert to BCD-4 function is used to output the 4-digit BCD equivalent of signed integer data. The original data is not changed by this function.

Data can be converted to BCD format to drive BCD-encoded LED displays or presets to external devices such as high-speed counters.

When the function receives power flow, it performs the conversion, making the result available via output Q. The function passes power flow when power is received, unless the specified conversion would result in a value that is outside the range 0 to 9999.



Parameters:

Parameter	Description
enable	When the function is enabled, the conversion is performed.
IN	IN contains a reference for the integer value to be converted to BCD-4.
ok	The ok output is energized when the function is performed without error.
Q	Output Q contains the BCD-4 form of the original value in IN.

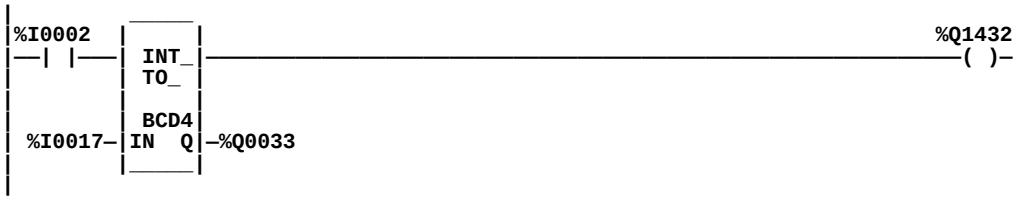
Valid Memory Types:

Parameter	flow	%I	%Q	%M	%T	%S	%G	%R	%AI	%AQ	const	none
enable	•											
IN		•	•	•	•		•	•	•	•	•	
ok	•											•
Q		•	•	•	•		•	•	•	•		

• Valid reference or place where power may flow through the function.

Example:

In the following example, whenever input %I0002 is set and no errors exist, the integer at input location %I0017 through %I0032 is converted to four BCD digits, and the result is stored in memory locations %Q0033 through %Q0048. Coil %Q1432 is used to check for successful conversion.



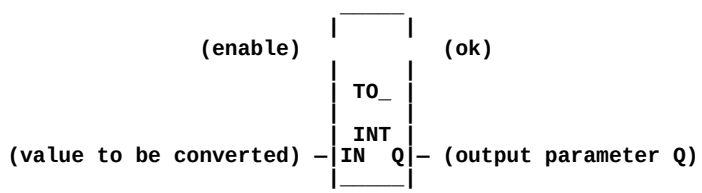
—>INT (BCD-4, REAL)

The Convert to Signed Integer function is used to output the integer equivalent of BCD-4 or REAL data. The original data is not changed by this function.

Note

The REAL data type is only available on 350 and 360 series CPUs, Release 9 or later, or on all releases of CPU352.

When the function receives power flow, it performs the conversion, making the result available via output Q. The function always passes power flow when power is received, unless the data is out of range.



Parameters:

Parameter	Description
enable	When the function is enabled, the conversion is performed.
IN	IN contains a reference for the BCD-4, REAL, or Constant value to be converted to integer.
ok	The ok output is energized whenever enable is energized, unless the data is out of range or NaN (Not a Number).
Q	Output Q contains the integer form of the original value in IN.

Valid Memory Types:

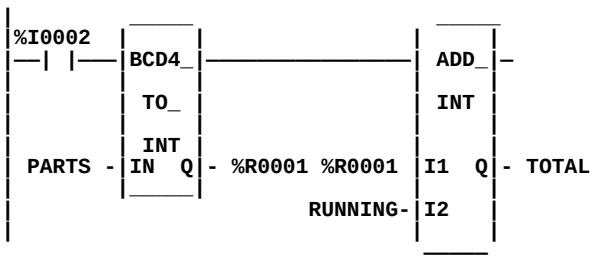
Parameter	flow	%I	%Q	%M	%T	%S	%G	%R	%AI	%AQ	const	none
enable	•											
IN		•	•	•	•		•	•	•	•	•	
ok	•											•
Q		•	•	•	•		•	•	•	•		

Note: For REAL data, the only valid types are %R, %AI, and %AQ.

• Valid reference or place where power may flow through the function.

Example:

In the following example, whenever input %I0002 is set, the BCD-4 value in PARTS is converted to a signed integer and passed to the ADD function, where it is added to the signed integer value represented by the reference RUNNING. The sum is output by the ADD function to the reference TOTAL.



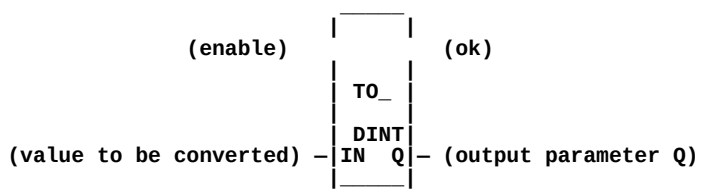
—>DINT (REAL)

The Convert to Double Precision Signed Integer function is used to output the double precision signed integer equivalent of real data. The original data is not changed by this function.

Note

The REAL data type is only available on 350 and 360 series CPUs, Release 9 or later, or on all releases of CPU352.

When the function receives power flow, it performs the conversion, making the result available via output Q. The function always passes power flow when power is received, unless the real value is out of range.



Parameters:

Parameter	Description
enable	When the function is enabled, the conversion is performed.
IN	IN contains a reference for the value to be converted to double precision integer.
ok	The ok output is energized whenever enable is energized, unless the real value is out of range.
Q	Q contains the double precision signed integer form of the original value in IN.

Note

It is possible for a loss of precision to occur when converting from REAL to DINT since the REAL has 24 significant bits.

Valid Memory Types:

Parameter	flow	%I	%Q	%M	%T	%S	%G	%R	%AI	%AQ	const	none
enable	•											
IN		o	o	o	o		o	•	•	•	•	
ok	•											•
Q								•	•	•		

• Valid reference or place where power may flow through the function.

Example:

In the following example, whenever input %I0002 is set, the real value at input location %R0017 is converted to a double precision signed integer, and the result is placed in location %R0001. The output %Q1001 is set whenever the function executes successfully.

—>REAL (INT, DINT, BCD-4, WORD)

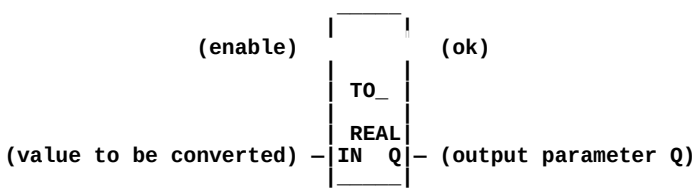
The Convert to Real function is used to output the real value of the input data. The original data is not changed by this function.

When the function receives power flow, it performs the conversion, making the result available via output Q. The function passes power flow when power is received, unless the specified conversion would result in a value that is out of range.

It is possible for a loss of precision to occur when converting from DINT to REAL since the number of significant bits is reduced to 24.

Note

This function is only available on 350 and 360 series CPUs, Release 9 or later, or on all releases of CPU352.



Parameters:

Parameter	Description
enable	When the function is enabled, the conversion is performed.
IN	IN contains a reference for the integer value to be converted to REAL.
ok	The ok output is energized when the function is performed without error.
Q	Q contains the REAL form of the original value in IN.

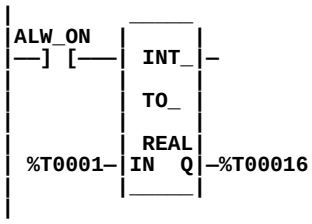
Valid Memory Types:

Parameter	flow	%I	%Q	%M	%T	%S	%G	%R	%AI	%AQ	const	none
enable	•											
IN		o	o	o	o		o	•	•	•	•	
ok	•											•
Q								•	•	•		

• Valid reference or place where power may flow through the function.
o Not valid for DINT_TO_REAL.

Example:

In the following example, the integer value of input IN is 678. The result value placed in %T0016 is 678.000.



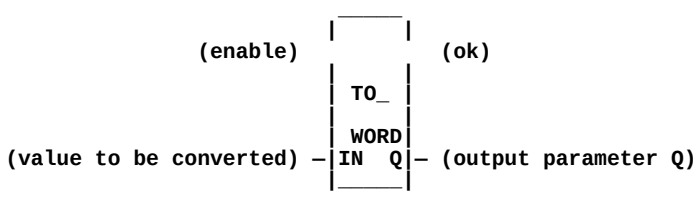
—>WORD (REAL)

The Convert to WORD function is used to output the WORD equivalent of real data. The original data is not changed by this function.

Note

This function is only available on the 350 and 360 series CPU.

When the function receives power flow, it performs the conversion, making the result available via output Q. The function passes power flow when power is received, unless the specified conversion would result in a value that is outside the range 0 to FFFFh.



Parameters:

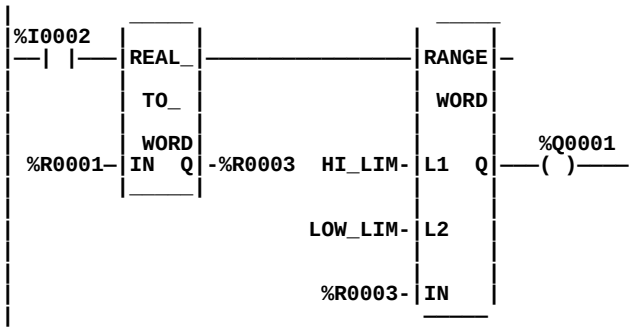
Parameter	Description
enable	When the function is enabled, the conversion is performed.
IN	IN contains a reference for the value to be converted to WORD.
ok	The ok output is energized when the function is performed without error.
Q	Q contains the unsigned integer form of the original value in IN.

Valid Memory Types:

Parameter	flow	%I	%Q	%M	%T	%S	%G	%R	%AI	%AQ	const	none
enable	•											
IN								•	•	•	•	
ok	•											•
Q		•	•	•	•		•	•	•	•		

• Valid reference or place where power may flow through the function.

Example:



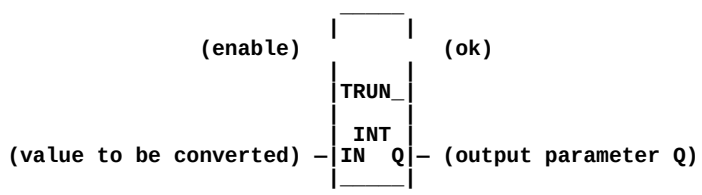
TRUN (INT, DINT)

The Truncate function is used to round the real number toward zero. The original data is not changed by this function.

Note

The 350 and 360 series CPUs (Release 9 or later and all releases of CPU352) are the only Series 90-30 CPUs with floating point capability; therefore, the TRUN function has no applicability for other 90-30 CPUs.

When the function receives power flow, it performs the conversion, making the result available via output Q. The function passes power flow when power is received, unless the specified conversion would result in a value that is out of range or unless IN is NaN (Not a Number).



Parameters:

Parameter	Description
enable	When the function is enabled, the conversion is performed.
IN	IN contains a reference for the real value to be truncated.
ok	The ok output is energized when the function is performed without error, unless the value is out of range or IN is NaN.
Q	Q contains the truncated INT or DINT value of the original value in IN.

Note

It is possible for a loss of precision to occur when converting from REAL to DINT since the REAL has 24 significant bits.

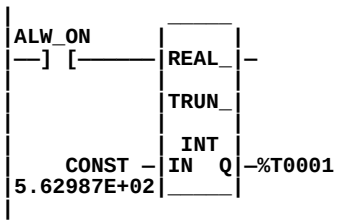
Valid Memory Types:

Parameter	flow	%I	%Q	%M	%T	%S	%G	%R	%AI	%AQ	const	none
enable	•											
IN								•	•	•	•	
ok	•											•
Q		o	o	o	o		o	•	•	•		

• Valid reference or place where power may flow through the function.
o Valid for REAL_TRUN_INT only.

Example:

In the following example, the displayed constant is truncated and the integer result 562 is placed in %T0001.



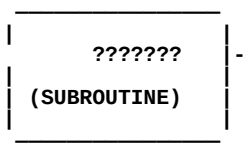
Section 9: Control Functions

This section describes the control functions, which may be used to limit program execution and alter the way the CPU executes the application program. (Refer to Chapter 2, section 1, "PLC Sweep Summary," for information on the CPU sweep.

Function	Description	Page
CALL	Causes program execution to go to a specified subroutine block.	4-108
DOIO	Services for one sweep a specified range of inputs or outputs immediately. (All inputs or outputs on a module are serviced if any reference locations on that module are included in the DO I/O function. Partial I/O module updates are not performed.) Optionally, a copy of the scanned I/O can be placed in internal memory, rather than the real input points.	4-109
SER	Sequential Event Recorder— collects data based on an event trigger. A function control block contains user-supplied information about function block execution, channel descriptions and operation parameters.	4-114
END	Provides a temporary end of logic. The program executes from the first rung to the last rung or the END instruction, whichever is encountered first. This instruction is useful for debugging purposes, but it is not permitted in SFC programming (refer to the Note on page 4-114).	4-123
MCR and MCRN	Programs a Master Control Relay. An MCR causes all rungs between the MCR and its subsequent ENDMCR to be executed without power flow. Logicmaster 90-30/20/Micro software supports two forms of the MCR function, a non-nested form (MCR) and a nested form (MCRN).	4-124
ENDMCR and ENDMCRN	Indicates that the subsequent logic is to be executed with normal power flow. Logicmaster 90-30/20/Micro software supports two forms of the ENDMCR function, a non-nested form (ENDMCR) and a nested form (ENDMCRN).	4-127
JUMP and JUMPN	Causes program execution to jump to a specified location (indicated by a LABEL, see below) in the logic. Logicmaster 90-30/20/Micro software supports two forms of the JUMP function, a non-nested form (JUMP) and a nested form (JUMPN).	4-128
LABEL and LABELN	Specifies the target location of a JUMP instruction. Logicmaster 90-30/20/Micro software supports two forms of the LABEL function, a non-nested form (LABEL) and a nested form (LABELN).	4-130
COMMENT	Places a comment (rung explanation) in the program. After programming the instruction, the text can be typed in by "zooming" into the instruction.	4-131
SVCREQ	Requests one of the following special PLC services: • Change/Read Task State and Number of Words to Checksum. • Change/Read Time-of-Day Clock. • Shut Down the PLC. • Clear Fault Tables. • Read Last-Logged Fault Table Entry. • Read Elapsed Time Clock. • Read I/O Override Status. • Read Master Checksum • Interrogate I/O • Read Elapsed Power Down Time	4-132
PID	Provides two PID (proportional/integral/derivative) closed-loop control algorithms: • Standard ISA PID algorithm (PIDISA). • Independent term algorithm (PIDIND).	4-165

CALL

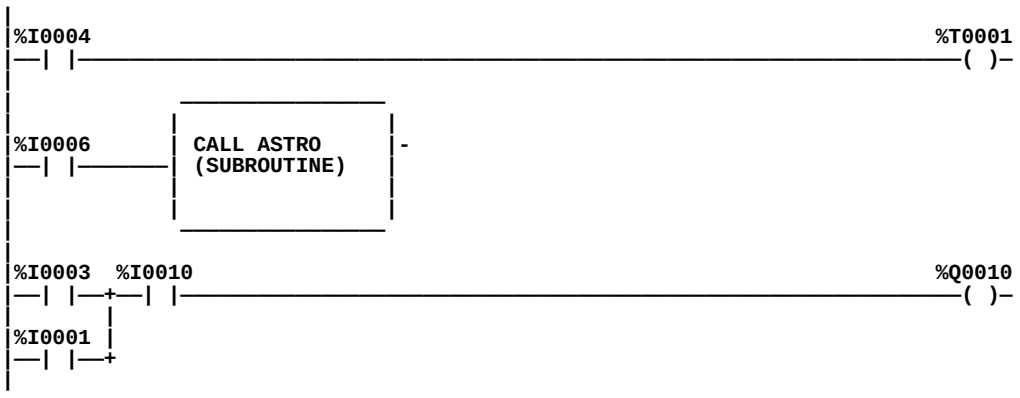
Use the CALL function to cause program execution to go to a specified subroutine block.



When the CALL function receives power flow, it causes the scan to go immediately to the designated subroutine block and execute it. After the subroutine block execution is complete, control returns to the point in the logic immediately following the CALL instruction.

Example:

The following example screen shows the subroutine CALL instruction as it appears in the calling block. By positioning the cursor within the instruction, you can press **F10** to zoom into the subroutine.



Note

Micro PLCs do not accommodate subroutines; therefore, the CALL function is inappropriate for use with a Micro PLC.

DOIO

The DO I/O (DOIO) function is used to update inputs or outputs for one scan while the program is running. The DOIO function can also be used to update selected I/O during the program in addition to the normal I/O scan.

If input references are specified, the function allows the most recent values of inputs to be obtained for program logic. If output references are specified, DO I/O updates outputs based on the most current values stored in I/O memory. I/O is serviced in increments of entire I/O modules; the PLC adjusts the references, if necessary, while the function executes.

The DOIO function has four input parameters and one output parameter. When the function receives power flow and input references are specified, the input points at the starting reference (ST) and ending at END are scanned. If a reference is specified for ALT, a copy of the new input values is placed in memory, beginning at that reference, and the real input points are not updated. ALT must be the same size as the reference type scanned. If a discrete reference is used for ST and END, then ALT must also be discrete. If no reference is specified for ALT, the real input points are updated.

When the DOIO function receives power flow and output references are specified, the output points at the starting reference (ST) and ending at END are written to the output modules. If outputs should be written to the output modules from internal memory, other than %Q or %AQ, the beginning reference can be specified for ALT. The range of outputs written to the output modules is specified by the starting reference (ST) and the ending reference (END).

Execution of the function continues until either all inputs in the selected range have reported, or all outputs have been serviced on the I/O cards. Program execution then returns to the next function following the DO I/O.

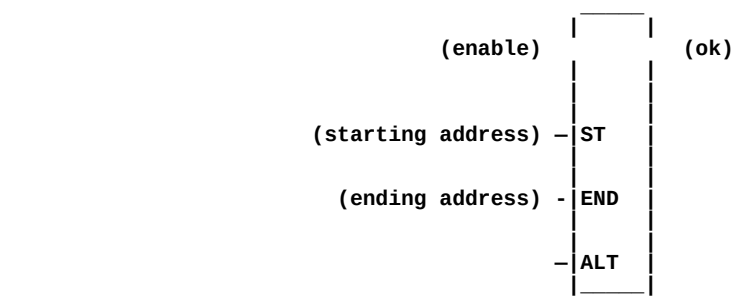
If the range of references includes an option module (HSC, APM, etc.), then all of the input data (%I and %AI) or all of the output data (%Q and %AQ) for that module will be scanned. The ALT parameter is ignored while scanning option modules. Also, the reference range must not include an Enhanced GCM module (see Note below).

Note

For Release 9.0 and later CPUs, the DOIO function ***can*** be used with an Enhanced GCM module.

The function passes power to the right whenever power is received, unless:

- Not all references of the type specified are present within the selected range.
- The CPU is not able to properly handle the temporary list of I/O created by the function.
- The range specified includes I/O modules that are associated with a “Loss of I/O” fault.



Parameters:

Parameter	Description
enable	When the function is enabled, a limited input or output scan is performed.
ST	ST is the starting address or set of input or output points or words to be serviced.
END	END is the ending address or set of input or output points or words to be serviced.
ALT	For the input scan, ALT specifies the address to store scanned input point/word values. For the output scan, ALT specifies the address to get output point/word values from to send to the I/O modules. For Model 331 and later CPUs, the ALT parameter can have an effect on speed of DOIO function block execution (see Note below and the section on the enhanced DO I/O function for 331 and later CPUs on page 4-110).
ok	The ok output is energized when the input or output scan completes normally.

Note

For Model 331 and later CPUs, the ALT parameter of the DOIO function block can be used to enter the slot of a single module in the main rack. When that is done, the DOIO function block will execute in 80 microseconds instead of the 236 microseconds required when the block is programmed without the ALT parameter. No error checking is performed to prevent overlapping reference addresses or module type mismatches.

Valid Memory Types:

Parameter	flow	%I	%Q	%M	%T	%S	%G	%R	%AI	%AQ	const	none
enable	•											
ST		•	•						•	•		
END		•	•						•	•		
ALT		•	•	•	•		•	•	•	•		•
ok	•											•

• Valid reference or place where power may flow through the function.

Input Example 1:

In the following example, when the enabling input %I0001 is ON, references %I0001 through %I0064 are scanned and %Q0001 is turned on. A copy of the scanned inputs is placed in internal memory from reference %M0001 through %M0064. The real input points are not updated. This form of the function can be used to compare the current values of input points with the values of input points at the beginning of the scan.

Input Example 2:

In the following example, when the enabling input %I0001 is ON, references %I0001 through %I0064 are scanned and %Q0001 is turned on. The scanned inputs are placed in the input status memory from reference %I0001 to %I0064. This form of the function allows input points to be scanned one or more times during the program execution portion of the CPU sweep.

Output Example 1:

In the following example, when the enabling input %I0001 is ON, the values at references %R0001 through %R0004 are written to analog output channels %AQ001 through %AQ004 and %Q0001 is turned on. The values at %AQ001 through %AQ004 are not written to the analog output modules.

Output Example 2:

In the following example, when the enabling input %I0001 is ON, the values at references %AQ001 through %AQ004 are written to analog output channels %AQ001 through %AQ004 and %Q0001 is turned on.

Enhanced DO I/O Function for 331 and Later CPUs

Caution

If the Enhanced DO I/O function is used in a program, the program should not be loaded by a version of Logicmaster 90-30/20 software prior to 4.01.

An enhanced version of the DO I/O (DOIO) function is available for Release 4.20, or later, of Models 331 and later CPUs. This enhanced version of the DOIO function can only be used on a single discrete input or discrete output 8-point, 16-point, or 32-point module.

The ALT parameter identifies the slot in the main rack that the module is located in. For example, a constant value of 2 in this parameter indicates to the CPU that it is to execute the enhanced version of the DOIO function block for the module in slot 2.

Note

The only checking done by the enhanced DOIO function block is to check the state of the module in the slot specified to see if the module is okay.

The enhanced DOIO function only applies to modules located in the main rack. Therefore, the ALT parameter must be between 2 and 5 for a 5-slot rack or 2 and 10 for a 10-slot rack.

The start and end references must be either %I or %Q. These references specify the first and last reference the module is configured for. For example, if a 16-point input module is configured at %I0001 through %I0016 in slot 10 of a 10-slot main rack, the ST parameter must be %I0001, the END parameter must be %I0016, and the ALT parameter must be 10, as shown below:

The following table compares the execution times of a normal DOIO function block for an 8-point, 16-point, or 32-point discrete input/output module with those of an enhanced DOIO function block.

Module	Normal DOIO Execution Time	Enhanced DOIO Execution Time
8-Pt Discrete Input Module 8-Pt Discrete Output Module	224 microseconds 208 microseconds	67 microseconds 48 microseconds
16-Pt Discrete Input Module 16-Pt Discrete Output Module	224 microseconds 211 microseconds	68 microseconds 47 microseconds
32-Pt Discrete Input Module 32-Pt Discrete Output Module	247 microseconds 226 microseconds	91 microseconds 50 microseconds

SER

The SER function (Sequential Event Recorder) function is used to collect data based on an event trigger. A function control block contains user-supplied information about function block execution, channel descriptions and operation parameters.

The function has a function control block, three input parameters and one output parameter, defined as follows:

Parameters:

Parameter	Description
enable	Whenever the function is enabled and the reset input is off, the SER function block will collect one sample from all configured channels.
Control Block	The 78-word array begins at the Word reference you specify here, and is used to define how the SER function will record data.
R	When the reset input receives power flow, the SER function will be reset regardless of the state of the enable input. The function block will remain in the reset state until power flow is removed from the reset input. The OK output will be turned off while in the reset state. When the power flow is removed from the reset input, channel sampling will resume.
T	When the trigger input receives power flow and the reset input is off, the SER moves to the triggered state and records the Trigger Time, Trigger Sample Offset and a sample. The trigger input requires power flow to the enable input so that a data sample may be collected from all configured channels on a trigger condition. The trigger sample will be recorded regardless of the number of samples taken. Once triggered, the event recorder will continue sampling until the Number of Samples After Trigger is satisfied. At which time it will stop collecting samples until power flow is seen on the reset input
ok	The ok output is energized whenever the trigger conditions are satisfied (specified by the Trigger Mode parameter), and all sampling is complete. The output will continue to receive power flow regardless of the state of the enable input until the reset receives power flow.

Note

This function requires version 9.00 or higher CPU firmware, and is available ***only*** on 350 and higher CPUs.

Valid Memory Types:

Parameter	flow	%I	%Q	%M	%T	%S	%G	%R	%AI	%AQ	const	none
enable	•											
Control Block								•				
R	•											
T	•											
ok	•											•

• Valid reference or place where power may flow through the function.

The Sequential Event Recorder function block is a 78-word array defining information about the data capture and trigger mechanism for the SER function. Perform these steps to configure parameters for the SER function block.

1. Set up the stored values for the array as defined in the table below. You can use block moves to initialize the registers, or initialize the data in the register table and store the table prior to activating the SER function.
2. Add the SER function to your ladder.

Parameter(Offset)	Description
Status(0)	Read only variable which indicates the current state of the SER function block. Additional information is provided in Status Extra Data, Offset 1 in the SER control block. NOTE: If an error is detected in the Control Block, The status will be set to 6, the OK output will be cleared and no action will occur. Valid settings for Status include: 0 = Reset 1 = Inactive 2 = Active 3 = Triggered 4 = Complete 5 = Overrun Error 6 = Parameter error
Status Extra Data(1)	A read-only variable that provides additional state information about the SER function. Click on “Status Extra Data” valid settings for this parameter.
Trigger Mode(2)	Defines the enabling action for the trigger condition: To select the Trigger boolean input as the enabling condition, set this parameter to 0. To select Full buffer as the enabling condition, set this parameter to 1. If the trigger condition is enabled by power flow to the Trigger boolean input, the OK boolean output will not pass power flow until the Number of Samples After Trigger has been satisfied. If the trigger condition is enabled by a Full buffer, the OK boolean output will pass power flow when the user's buffer is full. The buffer size is set through the Number of Samples parameter.
Trigger Time Format(3)	Determines how the Trigger Time will be displayed. For BCD (Binary Coded Decimal) display, set this parameter to 0. For POSIX format display, set this parameter to 1.
Reserved (4—7)	Words 4 through 7 are reserved and should be set to zero(0)
Num of Channels(8)	Specifies the number of bits of data that will be sampled and returned to the sample buffer for each execution of the function block. Valid choices are 8, 16, 24 or 32 bits. The increment is in byte size (8 bits) and any unused channels must be configured with a null channel description.

Parameter(Offset)	Description
Num of Samples(9)	Specifies the sample buffer size, in bytes. Valid choices are 1 to 1024 samples.
Num Samp After Trig(10)	Specifies the number of samples that are stored in the sample buffer when the trigger condition becomes true. This parameter may be set to a value between 0 and (Number of Samples – 1). This parameter is valid only when the Trigger Mode is set to Trigger.
Input Mod Slot(11)	Specifies the location of the input module for data sampling (slot in the main rack). Note: The user is responsible for guaranteeing this slot physically contains an input module. A slot number of 0 disables scanning of an input module. When an input module is scanned its values are stored locally, and the values of the reference addresses configured for the module are not affected. To store values from the scanned input module into the data block sample buffer a channel description must be used. If the module is not present, or faulted, at the time of the scan the data returned will be zero. A fault will not be logged in the fault table if this occurs, fault indication will be left to the IO scanner
Data Blk Seg Sel(12)	Specifies the data type that the user has allocated for the Data Block. For example, if you wanted to begin at %R0100, you would enter 08 for offset 12 and 99 for offset 13. Valid settings for this parameter include: %R (08h), %AI (0Ah), %AQ (0Ch).
Data Blk Offset(13)	Specifies the data type offset for the Data Block Segment Selector. The data type offset is zero (0) based. The user is responsible for allocating enough memory for the entire data block
Chan. Desrip. (14—77)	Specifies the reference location (Segment Selector, Length and Offset) associated with a particular channel. There can be from 1 to 32 channel descriptions, depending upon the number of channels being sampled and data length. Data is returned in the order as defined in this section.
Chan. Seg. Selector	Entered as a hexadecimal value, this word defines both the segment selector and data length (in bits). MSB = Segment Selector. LSB = Data Length. The data length is useful for samples that are contiguous.
	The Segment Selector may be set to any discrete data type: %I (46h), %Q (48h), %M (4Ch), %T (4Ah), %G (56h), %S (54h), %SA (4Eh), %SB (50h), %SC (52h), Null Selector (FFh), and Input Module Selector (00h). The length parameter can range from 1 - 32, but the sum of all of the lengths must not be greater than the Number of Channels parameter. A length greater than one allows for multiple contiguous channels to be configured with a single channel description. The range of valid offsets is dependent upon the data type and length. The offset indicates the location within the data table or input module at which to sample. The offset value is zero-based
Chan Offset	Entered as a hexadecimal value, this word defines the BIT offset for the data type or input module specified in the Segment Selector. The offset is zero-based. The range for this parameter varies, depending on the Segment Selector (data type and length). The offset indicates the location within the data table or input module at which to sample.

Status Extra Data

The Status Extra Data provides additional state information for the SER function.

State	Description
Inactive State (1)	State between the Reset State and the Active State. No actions are performed in this state. The Boolean output is held to no power flow. Transition to the Active State occurs when the function block receives enable power flow.
Active State (2)	State after the Enable Boolean input has received power flow, but the function block is not reset, in error, or triggered. One sample is recorded for each execution when the function block is enabled. The Boolean output is held to no power flow. The Trigger condition (specified by the Trigger Mode parameter) is monitored and will cause transition to the Triggered State if conditions are true. If more than the "Number of Samples" have been taken then Status Extra Data will be set to 0x01, otherwise it will 0x00.
Triggered State (3)	State if the trigger condition defined by Trigger Mode is true. Additional Samples are taken depending upon the trigger mode and parameter settings. The Boolean output is held to no power flow. Transition to the Complete State will occur when all sampling is complete. If more than the "Number of Samples" have been taken then Status Extra Data will be set to 0x01, otherwise it will be 0x00.
Complete State (4)	State after all sampling is complete. The Boolean output receives power flow. Only transition to the Reset State is allowed. If more than the "Number of Samples" have been taken then Status Extra Data will be set to 0x01, otherwise it will be 0x00.
Overrun Error State (5)	State if the Control/Data Block exceeds the end of its memory type. The Boolean output is held to no power flow. Only transition to the Reset State is allowed. Status Extra Data has no significance and will be cleared to zero.
Parameter Error State (6)	State if there is an error in the user supplied operation parameters. The Boolean output is held to no power flow. Only transition to the Reset State is allowed. The Status Extra Data word contains the offset into the control block at which the parameter error occurred.
Status Error State (7)	State if the Status Parameter becomes invalid. The Boolean output is held to no power flow. Only transition to the Reset State is allowed. The invalid status value will be stored in the Status Extra Data location in the Control Block.
Reset State (0)	State when the reset Boolean receives power flow. Sample Buffer, Trigger Sample Offset, Trigger Time, and Current Sample Offset are all cleared to zero. The Boolean output is held to no power flow. Transition to the ***Inactive State*** occurs when the reset power flow is removed. Status Extra Data has no significance and will be cleared to zero.

SER Data Block

The SER Data Block contains the sample buffer, sample offsets, and trigger information. This information is supplied by the CPU and the user should only read from this data area. It is the users responsibility to allocate enough register space for the Data Block. The block format is as follows:

Offset	Parameter Description
0	Current sample offset number. References the location where the most recent sample was placed. The parameter is zero-based. Valid ranges are –1 to 1023 Register Location of Sample = (Num Bytes per Sample) * (Offset Parameter) + (Sample Buffer Starting Register)
1	Trigger sample offset number. References the storage location of the sample obtained when the trigger condition transitioned to the True state. The parameter is zero-based. Valid ranges are 0 to 1023. Register Location of Sample = (Num Bytes per Sample) * (Offset Parameter) + (Sample Buffer Starting Register) Note: This value is not valid until the trigger condition is met. This value is set to 0 when the SER function is reset (through the reset input).
2 through 5	Trigger Time: Indicates the time, according to the Time of Day clock within the PLC, that the trigger condition transitioned to the true state within the function block. The time value is displayed in BCD format (default) although the time may be displayed in POSIX format also. The format is determined by the *Trigger Time Format* parameter in the Control Block. This value is initialized to zero upon activation of the reset Boolean input
6 to end samp buff.	Sample Buffer. The area of memory that holds the data samples. This area is set to zero when the reset parameter is energized. The sample buffer size varies, depending on the number of channels and sample size. The sample buffer is a circular buffer – when the last location is written, the next sample will overwrite the sample in the first register. {end of sample buffer = 5 + ({[(# of samples to be taken) * (# of channels to be sampled / 8)] +1} / 2)

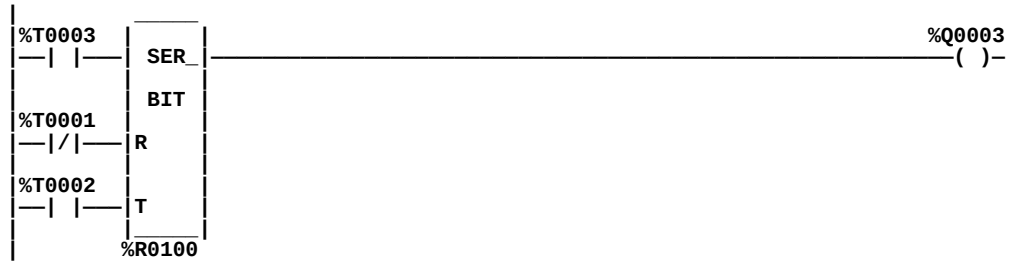
SER Notes

- The Control Block of the SER function block is scanned every time the function block is executed in the Reset, Active, or Triggered State. If the user changes one of the configuration parameters in the Control Block during program execution, the change will take effect the next time the SER function block associated with that Control Block is scanned. If an error is encountered, operation will be stopped and the function block will go to the appropriate error state. The user must correct the error and then reset the function block (enable the Reset input power flow) to begin sampling again.
- The SER function block must be reset (enable the Reset input power flow) before sampling is started. Resetting will initialize the data block area. If the function block status is not reset then it will execute with the current values in the data block. This will lead to the current sample offset being incorrect, and to invalid data in the data block.

- In a particular program, there can only be one Sequential Event Recorder function block that can be associated with each command and data block.
- If the user selects an input module to be scanned the PLC will NOT verify the module is a DISCRETE INPUT MODULE, or that any Channel Descriptions associated with the module have valid lengths and offsets based upon the module size. The user is responsible for correctly setting up the sampling of an Input Module.
- If an input module is selected to be scanned, it is only scanned once per function block execution. Multiple channel descriptions can target the input module, but the scanning is still only performed once per function block execution.
- If the user requires x channels where x is not equal to 8, 16, 24, but is less than 32 they must select a number of channels which is greater than x and a multiple of 8 and then fill in a null channel description for the remaining unused channels. A null channel description has a segment selector of 0xFFh, a length parameter which must equal the number of unused channels, and a 0 offset.
- The SER can be used in a periodic subroutine, however caution should be used when doing so. Depending on the mix of the samples being collected, the SER could take more than 1msec to execute and therefore it would not be practical to use it inside of a 1msec periodic subroutine. It will function exactly as any other function block does in the periodic subroutine, it is evaluated and executed according to the Boolean input logic.

Example:

In the following example, the offsets have been set up as described in the table below.



For the sake of example, assume that the system has a 16-point discrete input in rack 0 slot 4, has been executing for long enough that 572 samples (512 + 60) have been taken, and that the Enable boolean input is receiving power flow but the Reset and Trigger boolean inputs are not.

Offset	Register	Parameter Description	Value (dec)	Value (hex)
0	%R0100	Status:	2	0002
1	101	Status Extra Data:	1	0001
2	102	Trigger mode:	0	0000
3	103	Trigger Time Format:	0	0000
4	104	Reserved:	0	0000
5	105	Reserved:	0	0000
6	106	Reserved:	0	0000
7	107	Reserved:	0	0000
8	108	# of channels:	24	0018
9	109	# of samples to be taken:	512	0200
10	110	# of samples after trigger:	12	000C
11	111	Input module slot:	4	0004
12	112	Data Block Segment Selector:	8	0008
13	113	Data Block Offset:	200	00C8
14	114	Channel description 1: Seg. Sel. : Length	17921	4601
15	115	Offset	0	0000
16	116	Channel description 2: Seg. Sel. : Length	-253	FF03
17	117	Offset	0	0000
18	118	Channel description 3: Seg. Sel. : Length	3	0003
19	119	Offset	12	0012
20	120	Channel description 4: Seg. Sel. : Length	18434	4802
21	121	Offset	8	0008

Offset	Register	Parameter Description	Value (dec)	Value (hex)
22	122	Channel description 5: Seg. Sel. : Length	8	0008
23	123	Offset	0	0000
24	124	Channel description 6: Seg. Sel. : Length	-249	FF07
25	125	Offset	0	0000

The following is a description of the above control block.

- The status register is telling us that the FBK is in the Active state (2 - Active state). This means the function block is executing normally, and taking a sample each time the function block is encountered in program logic. The extra status data tells us that we have already taken more that 512 samples, and thus the sample buffer has already wrapped at least once.
- The event recorder is configured to trigger based on the Trigger boolean input.
- The reserved parameters are always set to 0.
- The user selected 24 channels of data with a sample buffer size of 512 samples. The sample buffer is not 512 bytes! It is 512 x (24/8) = 1536 bytes or 768 words.
- The number of samples that are to be gathered after the trigger is 12. (each sample is 3 bytes long)
- We are to scan the input module in rack 0 slot 4 so its current values are available for sampling.
- The data segment is 0x08 (registers) and the offset is 200 which places the start of the data block at %R0201. The offset is a zero-oriented value, but the register tables begin at %R0001. Therefore, the data block starting point is %R0001 + 200 = %R0201.
- The next section contains the channel descriptions. In this example 6 channel descriptions have been defined.

    1. The first channel description selects the %I Segment with a Length of 1, and offset of 0. This chooses %I0001 for channel 1.
    2. The second channel description selects the NULL Selector with Length of 3, and offset of 0. The NULL selector causes channels 2 - 4 to be ignored or “skipped”. These channels will always contain a sample value of Zero.
    3. The third channel description selects the Input Module Selector with a length of 3, and offset of 12. The Input Module Selector causes samples to be taken from the input module. This channel description chooses the values in points 13, 14, and 15 of the input module for channels 5 - 7.
    4. The fourth channel description selects the %Q Segment with a Length of 2, and offset of 8. This chooses %Q0009 and %Q0010 for channels 8 and 9.
    5. The fifth channel description is another Input Module Selector. It has a length of 8, and offset of 0. This causes the values for points 1 to 8 of the input module to be placed in channels 10 - 17.
    6. The sixth channel description is another NULL Selector. It has a Length of 7, and offset of 0. This NULL channel description causes channels 18 - 24 to be filled with Zeros. This

last channel description is required to pad the sample buffer out to the 24 bits specified in the number of channels parameter.

Since all 24 channels are configured there are no more channel descriptions needed and we have reached the end of our block.

The following table summarizes the values contained in a single sample based upon the above channel descriptions and control block:

Channel Number	Channel Contents
1	%I0001
2 - 4	Zeros
5	Input Module Point 13
6	Input Module Point 14
7	Input Module Point 15
8	%Q0009
9	%Q0010
10 - 17	Input Module Points 1 - 8
18 - 24	Zeros

Data Block

This is the format of the resulting data block from the control block described above. Note that it begins at register 201 as described by the segment offset parameters in the control block.

Offset	Register	Parameter Description	Value (dec)	Value (hex)
0	%R0201	Current sample offset #:	59	003B
1	202	Trigger sample offset #:	0	0000
2 - 5	203 – 206	Trigger time (BCD)	0 0 0 0	0000 0000 0000 0000
6 - 768	207 – 975	Samples Buffer:	sample data.	sample data.

Current sample offset is 59 meaning that we are 59 samples into the sample buffer (not 59 registers). With 3 bytes per sample we are actually at; 59 * 3 = 177 bytes or the hi byte of the 88th register. Since we have not met the trigger conditions yet the trigger sample and trigger time are 0 and the boolean output is not set. The sample buffer contains 512 samples where 59 is the newest sample and 60 is the oldest sample.

END

The END function provides a temporary end of logic. The program executes from the first rung to the last rung or the END function, whichever is encountered first.

The END function unconditionally terminates program execution. There can be nothing after the end function in the rung. No logic beyond the END function is executed, and control is transferred to the beginning of the program for the next sweep.

The END function is useful for debugging purposes because it prevents any logic which follows from being executed.

Logicmaster programming software provides an [END OF PROGRAM LOGIC] marker to indicate the end of program execution. This marker is used if no END function is programmed in the logic.

Example:

In the following example, an END is programmed to terminate the end of the current sweep.

```
|
|  STOP
|
|-[  END  ]
|
```

Note

Placing an END function in SFC logic or in logic called by SFC produces an "END Function Executed from SFC Action" fault in Release 7 or later CPUs. (In pre-Release 7 CPUs, it did not work correctly, but no Fault was generated.) For information about this fault, refer to the "System Configuration Mismatch" part of Chapter 3, Section 2.

MCR

All rungs between an active Master Control Relay (MCR) and its corresponding End Master Control Relay (ENDMCR) function are executed without power flow to coils. An ENDMCR function associated with the MCR is used to resume normal program execution. Unlike the JUMP instruction, MCRs can only occur in the forward direction. The ENDMCR instruction must appear after its corresponding MCR instruction in a program.

Logicmaster 90-30/20/Micro software supports two forms of the MCR function, a non-nested and a nested form. The non-nested form has been available since Release 1 of the software, and has the name MCR.

Note

The 350 and 360 series CPUs do not have the non-nested form, i.e., MCR. Use only the nested form, i.e., MCRN with 350 and 360 series CPUs.

There can be only one MCR instruction for each ENDMCR instruction. The range for non-nested MCRs and ENDMCRs cannot overlap the range of any other MCR/ENDMCR pair or any JUMP/LABEL pair of instructions. Non-nested MCRs cannot be within the scope of any other MCR/ENDMCR pair or any JUMP/LABEL pair. In addition, a JUMP/LABEL pair or an MCR/ENDMCR pair cannot be within the scope of an MCR/ENDMCR pair.

Note

The non-nested MCR function is the only Master Control Relay function that can be used in a Release 1 Series 90-30 PLC. The nested MCR function should be used for all new applications.

The nested form of the MCR function has the name MCRN, and is available in Release 2 and later releases of the Series 90-30 PLC. An MCRN function can be nested with other MCRN functions, provided they are nested correctly. An MCRN instruction and its corresponding ENDMCRN instruction must be contained completely within another MCRN/ENDMCRN pair.

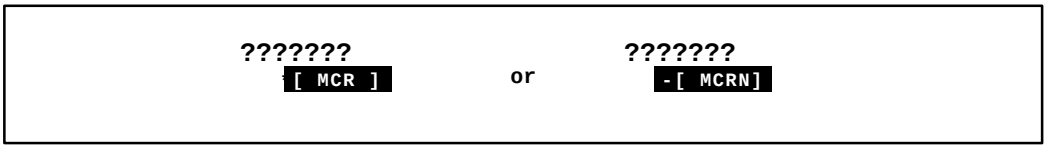
An MCRN function can be placed anywhere within a program, as long as it is properly nested with respect to other MCRNs, and does not occur in the range of any non-nested MCR or non-nested JUMP.

Note

Use only one (1) MCRN for each ENDMCRN with 350 and 360 series CPUs.

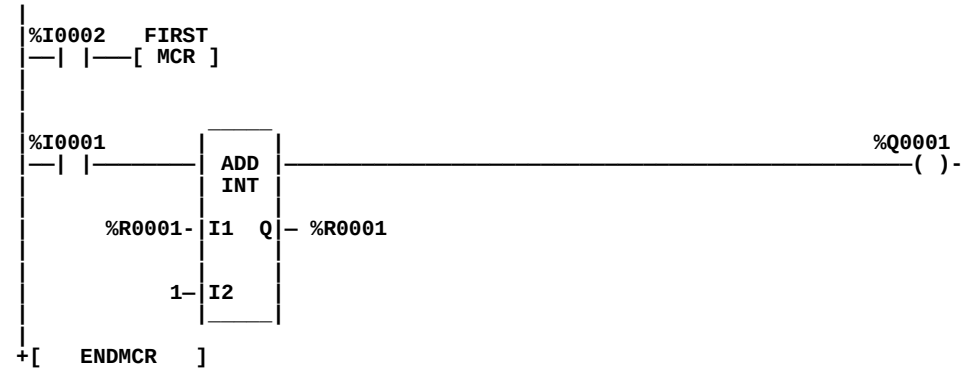
There can be multiple MCRN functions corresponding to a single ENDMCRN (except for the 350 and 360 series CPUs as noted above). This is analogous to the nested JUMP, where you can have multiple JUMPs to the same LABEL. For differences between the JUMP function and the MCR function, refer to the “Differences Between MCRs and Jumps” section on page 4-125.

Both forms of the MCR function have the same parameters. They both have an enable boolean input EN and also a name which identifies the MCR. This name is used again with an ENDMCR instruction. Neither the MCR nor the MCRN function has any outputs; there can be nothing after an MCR in a rung.

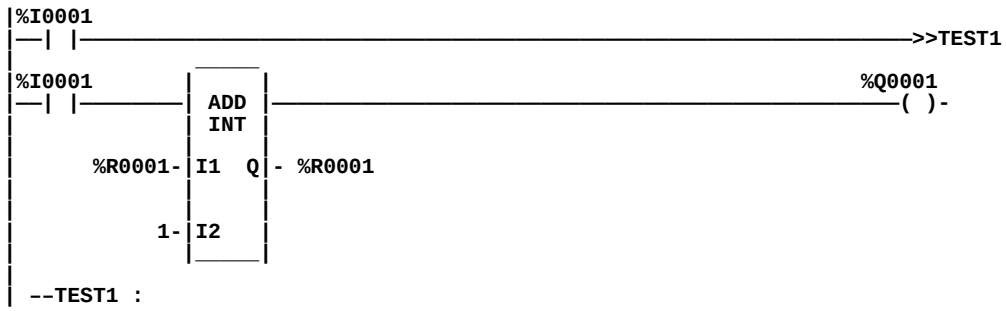


Differences Between MCRs and JUMPs

With an MCR function, function blocks within the scope of the MCR are executed ***without power flow***, and coils ***are turned off***. In the following example, when %I0002 is ON, the MCR is enabled. When the MCR is enabled—even if %I0001 is ON—the ADD function block is executed ***without*** power flow (i.e., it does not add 1 to %R0001), and %Q0001 is turned OFF.



With a JUMP function, any function blocks between the JUMP and the LABEL ***are not*** executed, and coils ***are not affected***. In the following example, when %I0002 is ON, the JUMP is taken. Since the logic between the JUMP and the LABEL is skipped, %Q0001 is unaffected (i.e., if it was ON, it remains ON; if it was OFF, it remains OFF).



Example:

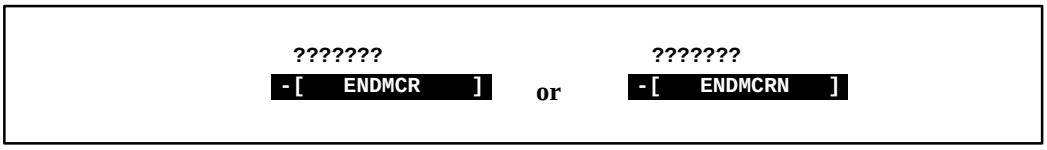
In the following example, whenever %I0002 allows power flow into the MCR function, program execution will continue without power flow to the coils until the associated ENDMCR is reached. If %I0001 and %I0003 are ON, %Q0001 is turned OFF and %Q0003 remains ON.

ENDMCR

Use the End Master Control Relay (ENDMCR) function to resume normal program execution after an MCR function. When the MCR associated with the ENDMCR is active, the ENDMCR causes program execution to resume with normal power flow. When the MCR associated with the ENDMCR is not active, the ENDMCR has no effect.

Logicmaster 90-30/20/Micro software supports two forms of the ENDMCR function, a non-nested and a nested form. The non-nested form, ENDMCR, must be used with the non-nested MCR function, MCR. The nested form, ENDMCRN, must be used with the nested MCR function, MCRN.

The ENDMCR function has a negated boolean input EN. The instruction enable must be provided by the power rail; execution cannot be conditional. The ENDMCR function also has a name, which identifies the ENDMCR and associates it with the corresponding MCR(s). The ENDMCR function has no outputs; there can be nothing before or after an ENDMCR instruction in a rung.



Example:

In the following examples, an ENDMCR instruction is programmed to terminate MCR range "clear."

Example of a non-nested ENDMCR:

```
|
|  CLEAR
|
|-[   ENDMCR    ]
|
```

Example of a nested ENDMCR:

```
|
|  CLEAR
|
|-[   ENDMCRN   ]
|
```

JUMP

Use the JUMP instruction to cause a portion of the program logic to be bypassed. Program execution will continue at the LABEL specified. When the JUMP is active, all coils within its scope are left at their previous states. This includes coils associated with timers, counters, latches, and relays.

Logicmaster 90-30/20/Micro software supports two forms of the JUMP instruction, a non-nested and a nested form. The non-nested form has been available since Release 1 of the software, and has the form ————————>>LABEL01, where LABEL01 is the name of the corresponding non-nested LABEL instruction.

For non-nested JUMPs, there can be only a single JUMP instruction for each LABEL instruction. The JUMP can be either a forward or a backward JUMP.

The range for non-nested JUMPs and LABELs cannot overlap the range of any other JUMP/LABEL pair or any MCR/ENDMCR pair of instructions. Non-nested JUMPs and their corresponding LABELs cannot be within the scope of any other JUMP/LABEL pair or any MCR/ENDMCR pair. In addition, an MCR/ENDMCR pair or another JUMP/LABEL pair cannot be within the scope of a non-nested JUMP/LABEL pair.

Note

The non-nested form of the JUMP instruction is the only JUMP instruction that can be used in a Release 1 Series 90-30 PLC. The nested JUMP function can be used (and is suggested for use) for all new applications.

Also, please note that the 350 and 360 series CPUs support only nested jumps. Non-nested jumps are not supported on 350 and 360 series CPUs.

The nested form of the JUMP instruction has the form ———N——>>LABEL01, where LABEL01 is the name of the corresponding nested LABEL instruction. It is available in Release 2 and later releases of Logicmaster 90-30/20/Micro software and PLC firmware.

A nested JUMP instruction can be placed anywhere within a program, as long as it does not occur in the range of any non-nested MCR or non-nested JUMP.

There can be multiple nested JUMP instructions corresponding to a single nested LABEL. Nested JUMPs can be either forward or backward JUMPs.

Both forms of the JUMP instruction are always placed in columns 9 and 10 of the current rung line; there can be nothing after the JUMP instruction in the rung. Power flow jumps directly from the instruction to the rung with the named label.

Non-nested JUMP:

Nested JUMP:

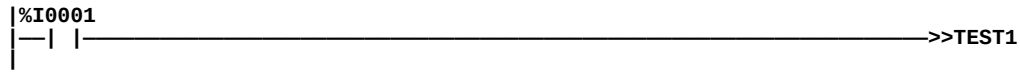
Caution

To avoid creating an endless loop with forward and backward JUMP instructions, a backward JUMP must contain a way to make it conditional.

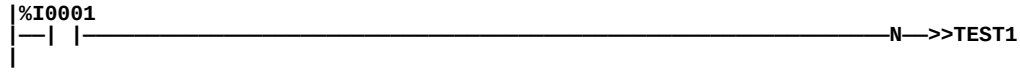
Example:

In the following examples, whenever JUMP TEST1 is active, power flow is transferred to LABEL TEST1.

Example of a non-nested JUMP:



Example of a nested JUMP:



LABEL

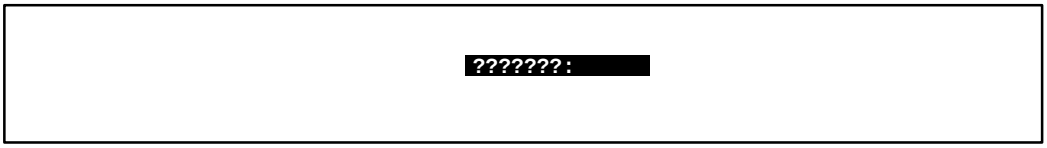
The LABEL instruction functions as the target destination of a JUMP. Use the LABEL instruction to resume normal program execution after a JUMP instruction.

There can be only one LABEL with a particular label name in a program. Programs without a matched JUMP/LABEL pair can be created and stored to the PLC, but cannot be executed.

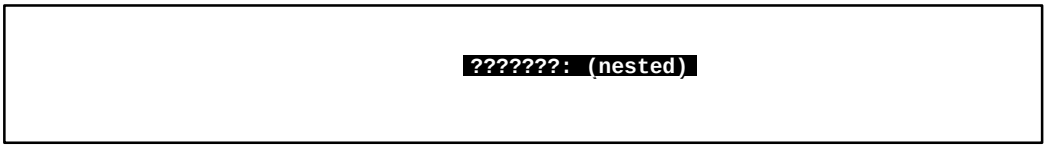
Logicmaster 90-30/20/Micro software supports two forms of the LABEL function, a non-nested and a nested form. The non-nested form, LABEL01:, must be used with the non-nested JUMP function, ————————>>LABEL01. The nested form, LABEL01:(nested), must be used with the nested JUMP function, ———N——>>LABEL01.

The LABEL instruction has no inputs and no outputs; there can be nothing either before or after a LABEL in a rung.

Non-nested LABEL:



Nested LABEL:



Example:

In the following examples, power flow from JUMP TEST1 is resumed, starting at LABEL TEST1.

Example of a non-nested LABEL:

```
TEST1  :
```

Example of a nested LABEL:

```
TEST1  :(nested)
```

COMMENT

Use the COMMENT function to enter a comment (rung explanation) in the program. A comment can have up to 2048 characters of text. It is represented in the ladder logic like this:

The text can be read or edited by moving the cursor to (* COMMENT *) after accepting the rung and selecting Zoom (F10). Comment text can also be printed.

Longer text can be included in printouts using an annotation text file, as described below:

1. Create the comment:
   A. Enter text to the point where the text from the other file should begin.
   B. Move the cursor to the beginning of a new line and enter **\I** or **\i**, the drive followed by a colon, the subdirectory or folder, and the file name, as shown in this example:

      ```
      \I d:\text\commnt1
      ```

      The drive designation is not necessary if the file is located on the same drive as the program folder.
   C. Continue editing the program, or exit to MS-DOS.
2. After exiting the programmer, create a text file using any MS-DOS compatible software package. Give the file the file name entered in the comment, and place it on the drive specified in the comment.

SVCREQ

Use the Service Request (SVCREQ) function to request one of the following special PLC services:

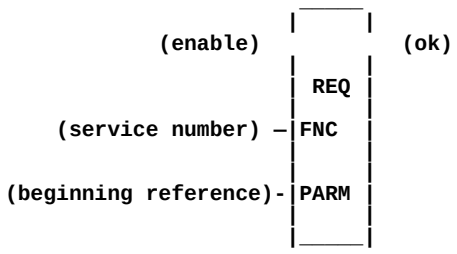
Table 4-3. Service Request Functions

Function	Description
1	Change/Read Constant Sweep Timer.
2	Read Window Values.
3	Change Programmer Communications Window Mode and Timer Value.
4	Change System Comm. Window Mode and Timer Value.
6	Change/Read Checksum Task State and Number of Words to Checksum.
7	Change/Read Time-of-Day Clock.
13	Shut Down the PLC.
14	Clear Fault Tables.
15	Read Last-Logged Fault Table Entry.
16	Read Elapsed Time Clock.
18	Read I/O Override Status.
23	Read Master Checksum
26/30	Interrogate I/O
29	Read Elapsed Power Down Time

The SVCREQ function has three input parameters and one output parameter. When the SVCREQ receives power flow, the PLC is requested to perform the function FNC indicated. Parameters for the function begin at the reference given for PARM. The SVCREQ function passes power flow unless an incorrect function number, incorrect parameters, or out-of-range references are specified. Additional causes for failure are described on the pages that follow.

The reference given for PARM may represent any type of word memory (%R, %AI, or %AQ). This reference is the first of a group that make up the “parameter block” for the function. Successive 16-bit locations store additional parameters. The total number of references required will depend on the type of SVCREQ function being used.

Parameter blocks may be used as both inputs for the function and the location where data may be output after the function executes. Therefore, data returned by the function is accessed at the same location specified for PARM.



Parameters:

Parameter	Description
enable	When enable is energized, the request service request is performed.
FNC	FNC contains the constant or reference for the requested service.
PARM	PARM contains the beginning reference for the parameter block for the requested service.
ok	The ok output is energized when the function is performed without error.

Valid Memory Types:

Parameter	flow	%I	%Q	%M	%T	%S	%G	%R	%AI	%AQ	const	none
enable	•											
FNC		•	•	•	•		•	•	•	•	•	
PARM		•	•	•	•		•	•	•	•		
ok	•											•

• Valid reference or place where power may flow through the function.

Example:

In the following example, when the enabling input %I0001 is ON, SVCREQ function number 7 is called, with the parameter block located starting at %R0001. Output coil %Q0001 is set ON if the operation succeeds.

SVCREQ #1: Change/Read Constant Sweep Timer

Beginning with 90-30 CPU Release 8, use SVCREQ function #1 to:

- Disable **CONSTANT SWEEP** mode.
- Enable **CONSTANT SWEEP** mode and use the old timer value.
- Enable **CONSTANT SWEEP** mode and use a new timer value.
- Set a new timer value only.
- Read **CONSTANT SWEEP** mode state and timer value.

Note

Of the CPUs discussed in this manual, Service Request 1 is supported *only* by 90-30 CPUs, beginning with Release 8.0.

The parameter block has a length of two words.

To disable **CONSTANT SWEEP** mode, enter SVCREQ function #1 with this parameter block:

0	address
ignored	address + 1

To enable **CONSTANT SWEEP** mode, enter SVCREQ function #1 with this parameter block:

1	address
0 or timer value	address + 1

Note

If the timer should use a new value, enter it in the second word. If the timer value should not be changed, enter 0 in the second word. If the timer value does not already exist, entering 0 will cause the function to set the OK output to OFF.

To change the timer value **without** changing the selection for sweep mode state, enter SVCREQ function #1 with this parameter block:

2	address
new timer value	address + 1

To read the current timer state and value without changing either, enter SVCREQ function #1 with this parameter block:

3	address
ignored	address + 1

Note

After using SVCREQ function #1 with the parameter block on the previous page, Release 8 and higher CPUs will provide the return values 0 for Normal Sweep, 1 for Constant Sweep. Do not confuse this with the *input* values shown below.

Successful execution will occur, unless:

1. A number other than 0, 1, 2, or 3 is entered as the requested operation:

0	Disable **CONSTANT SWEEP** mode.
1	Enable **CONSTANT SWEEP** mode.
2	Set a new timer value only.
3	Read **CONSTANT SWEEP** mode and timer value. (See Note above).

2. The time value is greater than 2550 ms (2.55 seconds).
3. Constant sweep time is enabled with no timer value programmed, or with an old value of 0 for the timer.

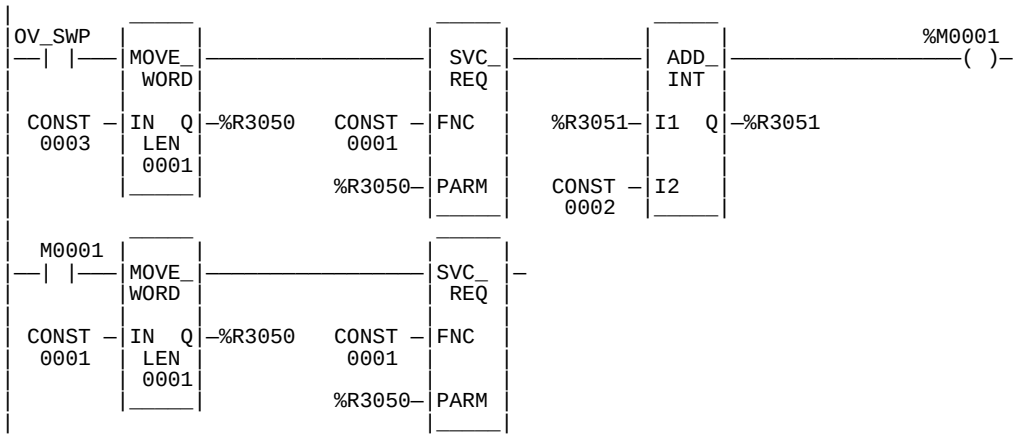
After the function executes, the function returns the timer state and value in the same parameter block references:

0 = disabled	
1 = enabled	address
current timer value	address + 1

If word address + 1 contains the hexadecimal value FFFF, no timer value has ever been programmed.

Example:

This example shows logic in a program block. When enabling contact OV_SWP is set, the constant sweep timer is read, the timer is increased by two milliseconds, and the new timer value is sent back to the PLC. The parameter block is in local memory at location %R3050. Because the MOVE and ADD functions require three horizontal contact positions, the example logic uses discrete internal coil %M0001 as a temporary location to hold the successful result of the first rung line. On any sweep in which OV_SWP is not set, %M0001 is turned off.



SVCREQ #2: Read Window Values

Use SVCREQ function #2 to obtain the current window mode time values for the programmer communications window, the system communications window, and the background task window.

Note

Of the CPUs discussed in this manual, Service Request 2 is supported *only* by 90-30 CPUs, beginning with Release 8.0.

There are three modes for each window:

Mode Name	Value	Description
Limited Mode	0	The execution time of the window is limited to its respective default value or to a value defined using SVCREQ function #3 for the programmer communications window or SVCREQ function #4 for the systems communications window. The window will terminate when it has no more tasks to complete.
Constant Mode	1	Each window will operate in a **`RUN TO COMPLETION`** mode, and the PLC will alternate among the three windows for a time equal to the sum of each window's respective time value. If one window is placed in **`CONSTANT`** mode, the remaining two windows are automatically placed in **`CONSTANT`** mode. If the PLC is operating in **`CONSTANT WINDOW`** mode and a particular window's execution time is not defined using the associated SVCREQ function, the default time for that window is used in the constant window time calculation.
Run to Completion Mode	2	Regardless of the window time associated with a particular window, whether default or defined using a service request function, the window will run until all tasks within that window are completed.

A window is disabled when the time value is zero.

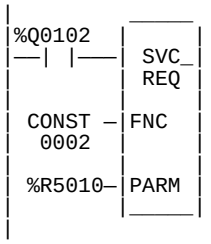
The parameter block has a length of three words:

	High Byte	Low Byte	
Programmer Window	Mode	Value in ms	address
System Communications Window	Mode	Value in ms	address + 1
Background Window			address + 2

All parameters are output parameters. It is not necessary to enter values in the parameter block to program this function. Output values for all three windows are given in milliseconds.

Example:

In the following example, when enabling output %Q0102 is set, the PLC operating system places the current time values of the three windows in the parameter block starting at location %R5010. Additional examples showing the Read Window Values function are included in the next three SYS REQ function descriptions.



SVCREQ #3: Change Programmer Communications Window Mode and Timer Value

Use SVCREQ function #3 to change the programmer communications window mode and timer value. The change will occur in the CPU sweep following the sweep in which the function is called.

Note

Of the CPUs discussed in this manual, Service Request 3 is supported *only* by 90-30 CPUs, beginning with Release 8.0.

The SVCREQ function #3 will pass power flow to the right unless a mode other than 0 (Limited), 1 (Constant), or 2 (Run-to-Completion) is selected.

The parameter block has a length of one word.

To disable the programmer window, enter SVCREQ function #3 with this parameter block:

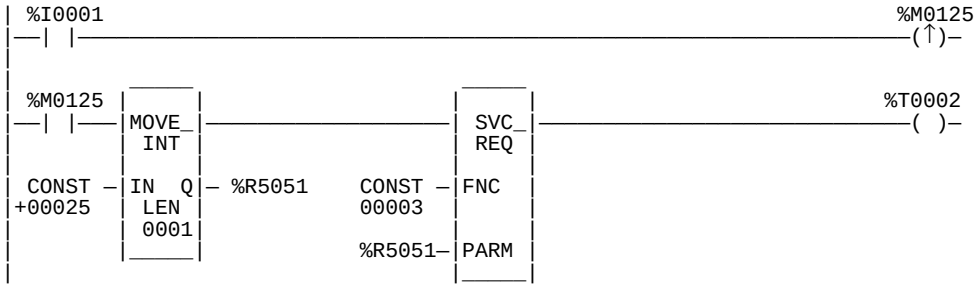
High Byte	Low Byte	
0	0	address

To enable the programmer window, enter SVCREQ function #3 with this parameter block:

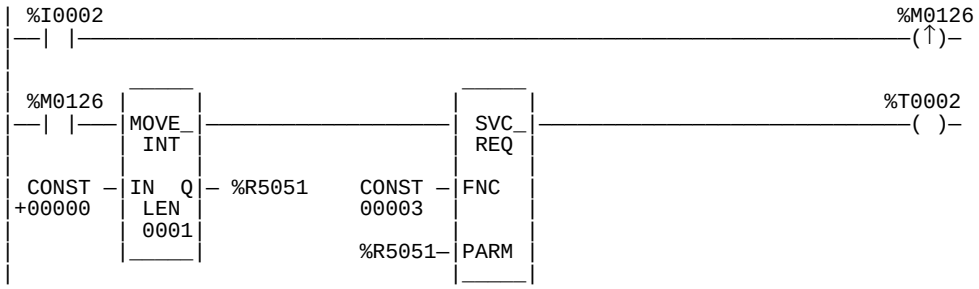
High Byte	Low Byte	
Mode	Value from 1 to 255 ms	address

Example:

In the following example, when %M0125 transitions on, the programmer communications window is enabled and assigned a value of 25 ms. The parameter block is in memory location %R5051.



To disable the programmer communications window, use Service Request 3 to assign a value of zero (0). In this example, when %M0126 transitions on, the programmer communications window is enabled and assigned a value of 0 ms. The parameter block is in memory location %R5051.



SVCREQ #4: Change System Comm. Window Mode and Timer Value

Use SVCREQ function #4 to change the system communications window mode and timer value. The change will occur in the CPU sweep following the sweep in which the function is called.

Note

Of the CPUs discussed in this manual, Service Request 4 is supported *only* by 90-30 CPUs, beginning with Release 8.0.

The SVCREQ function #4 will pass power flow to the right unless a mode other than 0 (Limited), 1 (Constant), or 2 (Run-to-Completion) is selected.

The parameter block has a length of one word.

To disable the system communications window, enter SVCREQ function #4 with this parameter block:

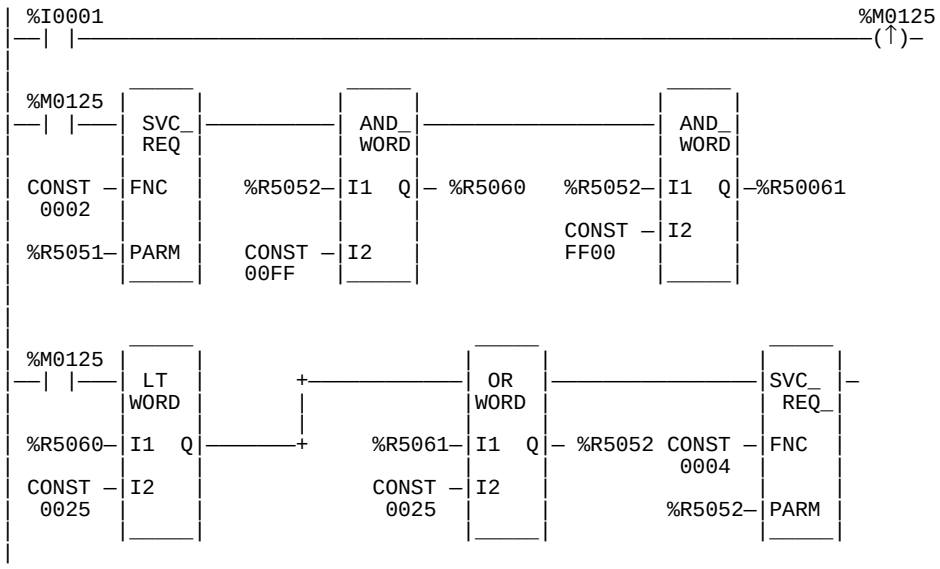
High Byte	Low Byte	
0	0	address

To enable the system communications window, enter SVCREQ function #4 with this parameter block:

High Byte	Low Byte	
Mode	Value from 1 to 255 ms	address

Example:

In the following example, when enabling output %M0125 transitions on, the mode and timer value of the system communications window is read. If the timer value is greater than or equal to 25 ms, the value is not changed. If it is less than 25 ms, the value is changed to 25 ms. In either case, when the rung completes execution the window is enabled. The parameter block for all three windows is at location %R5051. Since the mode and timer for the system communications window is the second value in the parameter block returned from the Read Window Values function (function #2), the location of the existing window time for the system communications window is in the low byte of %R5052.



SVCREQ #6: Change/Read Number of Words to Checksum

Use the SVCREQ function with function number 6 in order to:

- Read the current word count.
- Set a new word count.

Successful execution will occur, unless some number other than 0 or 1 is entered as the requested operation (see below).

For the Checksum Task functions, the parameter block has a length of 2 words.

To Read the Current Word Count:

Enter SVCREQ function 6 with this parameter block:

0	address
ignored	address + 1

After the function executes, the function returns the current checksum in the second word of the parameter block. No range is specified for the read function; the value returned is the number of words currently being checksummed.

0	address
current word count	address + 1

To Set a New Word Count:

Enter SVCREQ function 6 with this parameter block:

1	address
new word count	address + 1

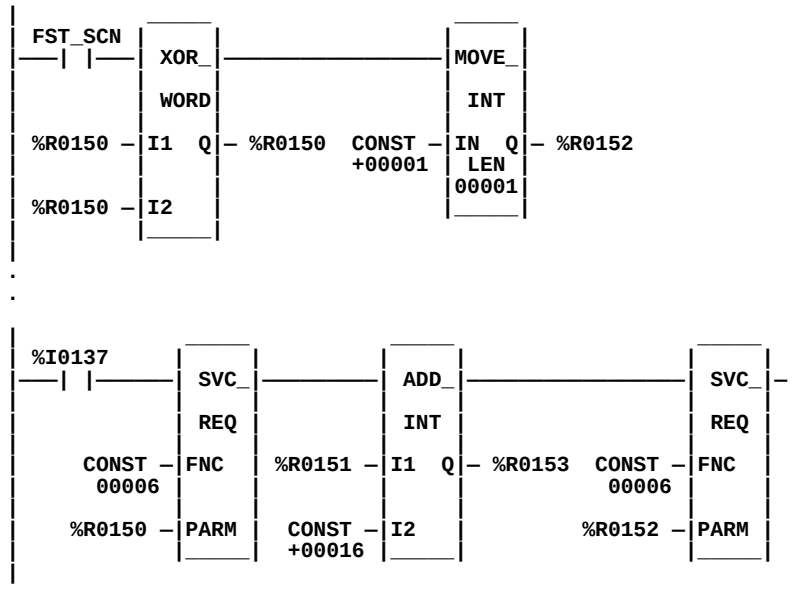
Entering 1 causes the PLC to adjust the number of words to be checksummed to the value given in the second word of the parameter block. For either the 331 or 311 CPU, the number can be either 0 or 32; in the 211 CPU, the value can be either 0 or 4.

Note

This Service Request is not available on Micro PLCs.

Example:

In the following example, when enabling contact FST_SCN is set, the parameter blocks for the checksum task function are built. Later in the program when input %I0137 turns on, the number of words being checksummed is read from the PLC operating system. This number is increased by 16, with the results of the ADD_INT function being placed in the “hold new count for set” parameter. The second service request block requests the PLC to set the new word count.



The example parameter blocks are located at address %R0150. They have the following content:

0 = read current count	%R0150
hold current count	%R0151

1 = set current count	%R0152
hold new count for set	%R0153

SVCREQ #7: Change/Read Time-of-Day Clock

Use the SVCREQ function with function number 7 to read and set the time-of-day clock in the PLC.

Note

This function is available only in 331 or higher 90-30 CPUs and on the 28-point Series 90 Micro PLC CPUs (that is, IC693UDR005, IC693UAA007, and IC693UDR010) and the 23-point Series 90 Micro PLC CPUs (IC693UAL006).

Successful execution will occur unless:

1. Some number other than 0 or 1 is entered as the requested operation (see below).
2. An invalid data format is specified.
3. The data provided is not in the expected format.

For the date/time functions, the length of the parameter block depends on the data format. BCD format requires 6 words; packed ASCII requires 12 words.

0 = read time and date 1 = set time and date	address
1 = BCD format 3 = packed ASCII format	address + 1
data	address + 2 to end

In word 1, specify whether the function should read or change the values.

```
0    =    read
1    =    change
```

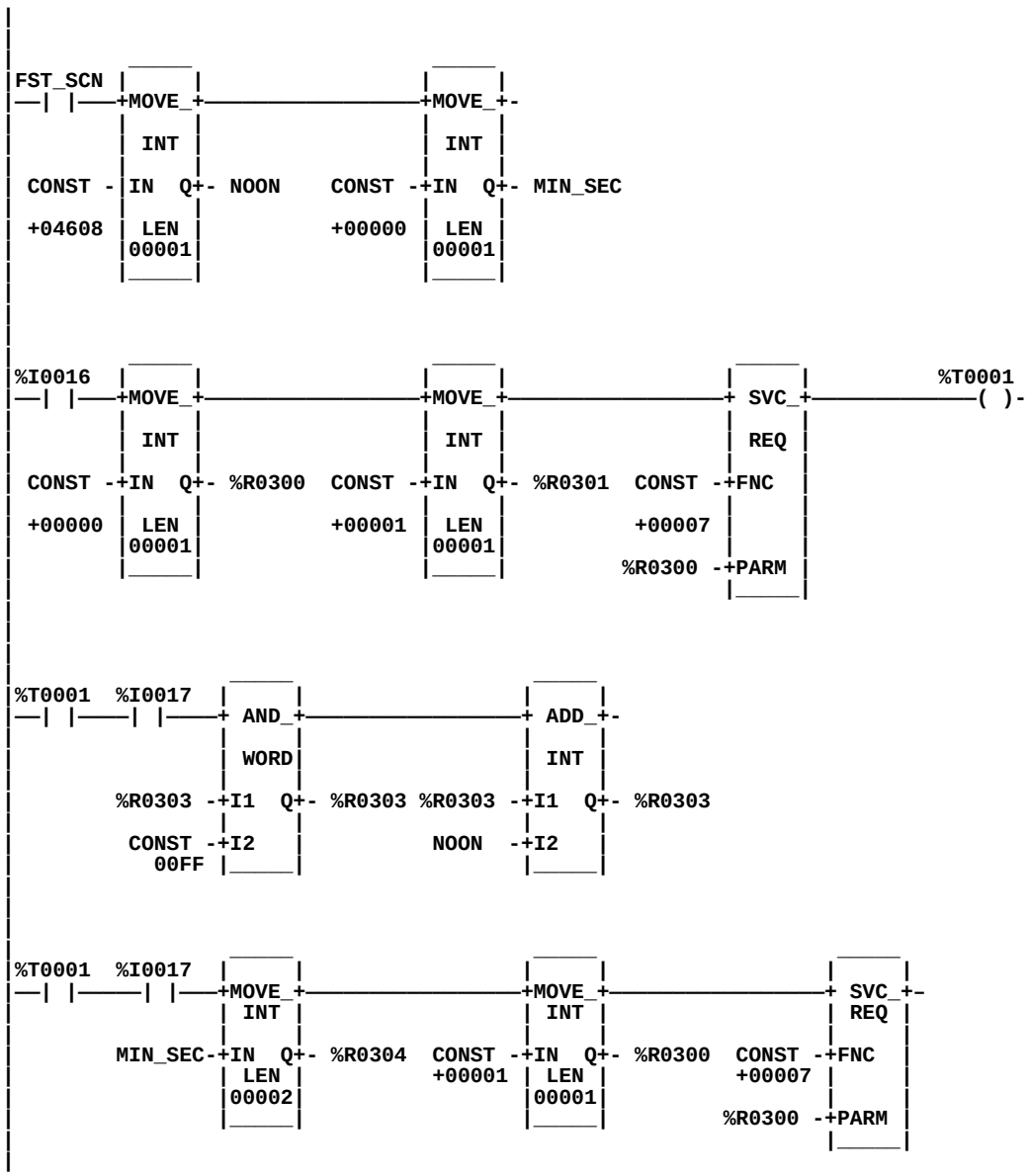
In word 2, specify a data format:

```
1    =    BCD
3    =    packed ASCII with embedded spaces and colons
```

Words 3 to the end of the parameter block contain output data returned by a read function, or new data being supplied by a change function. In both cases, format of these data words is the same. When reading the date and time, words (address + 2) through (address + 8) of the parameter block are ignored on input.

Example:

In the following example, when called for by previous logic, a parameter block for the time-of-day clock is built to first request the current date and time, and then set the clock to 12 noon using the BCD format. The parameter block is located at global data location %R0300. Array NOON has been set up elsewhere in the program to contain the values 12, 0, and 0. (Array NOON must also contain the data at %R0300.) The BCD format requires six contiguous memory locations for the parameter block.



Parameter Block Contents

Parameter block contents for the different data formats are shown on the following pages. For both data formats:

- Hours are stored in 24-hour format.
- Day of the week is a numeric value:

Value	Day of the Week
1	Sunday
2	Monday
3	Tuesday
4	Wednesday
5	Thursday
6	Friday
7	Saturday

To Change/Read Date and Time Using BCD Format:

In BCD format, each of the time and date items occupies a single byte. This format requires six words. The last byte of the sixth word is not used. When setting the date and time, this byte is ignored; when reading date and time, the function returns a null character (00).

High Byte	Low Byte		Example output parameter block: Read Date and Time in BCD format (Sun., July 3, 1988, at 2:45:30 p.m.)	
1 = change or	0 = read	address	0	
1		address + 1	1	
month	year	address + 2	07	88
hours	day of month	address + 3	14	03
seconds	minutes	address + 4	30	45
(null)	day of week	address + 5	00	01

To Change/Read Date and Time using Packed ASCII with Embedded Colons Format

In Packed ASCII format, each digit of the time and date items is an ASCII formatted byte. In addition, spaces and colons are embedded into the data to permit it to be transferred unchanged to a printing or display device. This format requires 12 words.

High Byte	Low Byte		Example output parameter block: Read Date and Time in Packed ASCII Format (Mon, Oct. 2, 1989 at 23:13:00)	
1 = change or	0 = read	address	0	
3		address + 1	3	
year	year	address + 2	39	38
month	(space)	address + 3	31	20
(space)	month	address + 4	20	30
day of month	day of month	address + 5	32	30
hours	(space)	address + 6	32	20
:	hours	address + 7	3A	33
minutes	minutes	address + 8	33	31
seconds	:	address + 9	30	3A
(space)	seconds	address + 10	20	30
day of week	day of week	address + 11	32	30

SVCREQ #8: Reset Watchdog Timer

Use SVCREQ function #8 to reset the watchdog timer during the sweep.

Note

Of the CPUs discussed in this manual, Service Request 8 is supported *only* by 90-30 CPUs, beginning with Release 8.0.

When the watchdog timer expires, the PLC shuts down without warning. This function allows the timer to keep going during a time-consuming task (for example, while waiting for a response from a communications line).

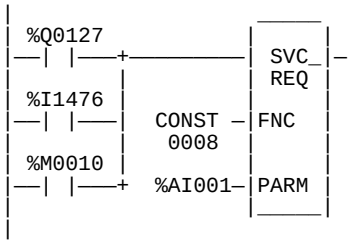
Caution

Be sure that restarting the watchdog timer does not adversely affect the controlled process.

This function has no associated parameter block; however, the programming software requires that an entry be made for PARM. Enter any appropriate reference here; it will not be used.

Example:

In the following example, when enabling output %Q0127 or input %I1476 or internal coil %M0010 is set, the watchdog timer is reset.



SVCREQ #9: Read Sweep Time from Beginning of Sweep

Use SVCREQ function #9 to read the time in milliseconds since the start of the sweep. The data is in 16-bit Word format.

Note

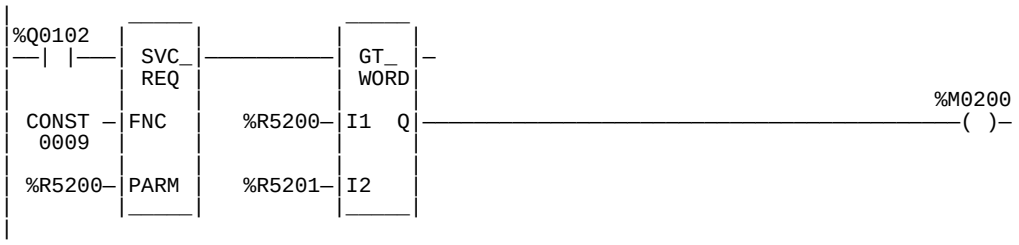
Of the CPUs discussed in this manual, Service Request 9 is supported *only* by 90-30 CPUs, beginning with Release 8.0.

The parameter block is an output parameter block only; it has a length of one word.

time since start of sweep	address

Example:

In the following example, the elapsed time from the start of the sweep is always read into location %R5200. If it is greater than the value in %R5201, internal coil %M0200 is turned on.



SVCREQ #10: Read Folder Name

Use SVCREQ function #10 to read the name of the currently-executing folder.

Note

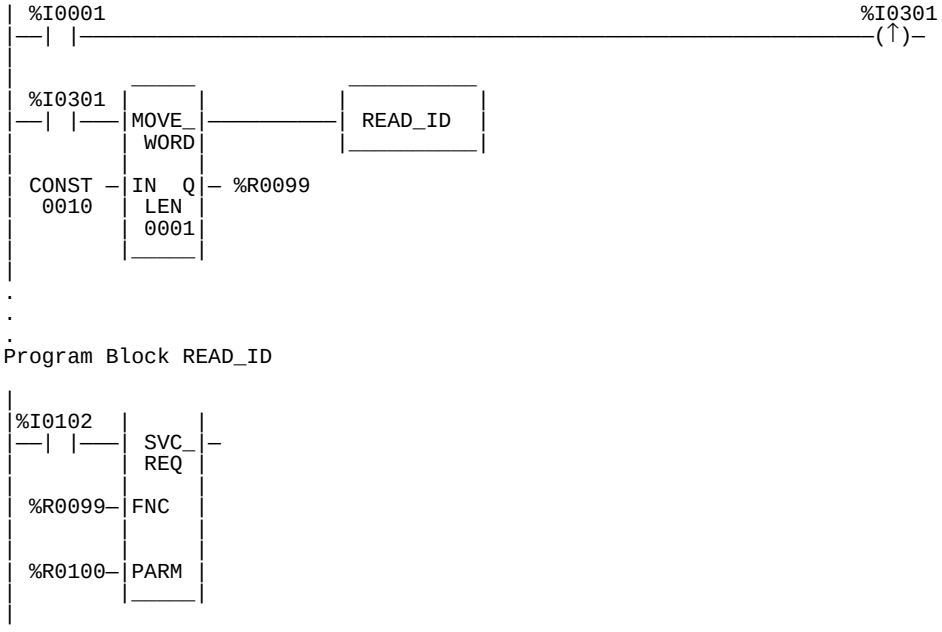
Of the CPUs discussed in this manual, Service Request 10 is supported *only* by 90-30 CPUs, beginning with Release 8.0.

The output parameter block has a length of four words. It returns eight ASCII characters; the last is a null character (00h). If the program name has fewer than seven characters, null characters are appended to the end.

Low Byte	High Byte	
character 1	character 2	address
character 3	character 4	address + 1
character 5	character 6	address + 2
character 7	00	address + 3

Example:

In the following example, when enabling input %I0301 transitions off, register location %R0099 is loaded with the value 10, which is the function code for the Read Folder Name function. The Program Block READ_ID is then called to actually retrieve the folder name. The parameter block is located at address %R0100. READ_ID is also used in the next example.



SVCREQ #11: Read PLC ID

Use SVCREQ function #11 to read the name of the Series 90 PLC executing the program.

Note

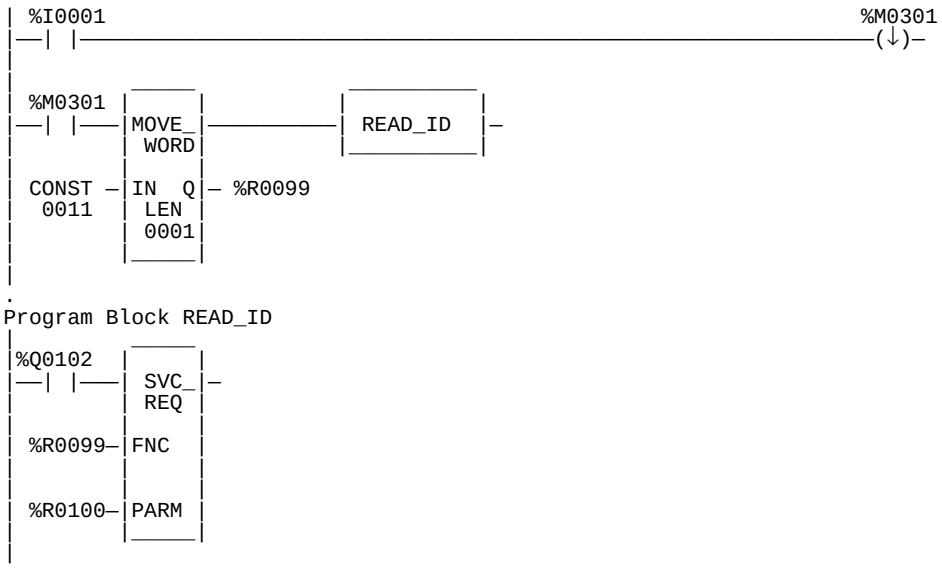
Of the CPUs discussed in this manual, Service Request 11 is supported *only* by 90-30 CPUs, beginning with Release 8.0.

The output parameter block has a length of four words. It returns eight ASCII characters; the last is a null character (00h). If the PLC ID has fewer than seven characters, null characters are appended to the end.

Low Byte	High Byte	
character 1	character 2	address
character 3	character 4	address + 1
character 5	character 6	address + 2
character 7	00	address + 3

Example:

In the following example, when enabling input %I0001 transitions off, register location %R0099 is loaded with the value 11, which is the function code for the Read PLC ID function. The program block READ_ID is then called to actually retrieve the ID. The parameter block is located at address %R0100. Except for the enabling contact and function number, this is the same code used in the previous example.



SVCREQ #12: Read PLC Run State

Use SVCREQ function #12 to read the current RUN state of the PLC CPU.

Note

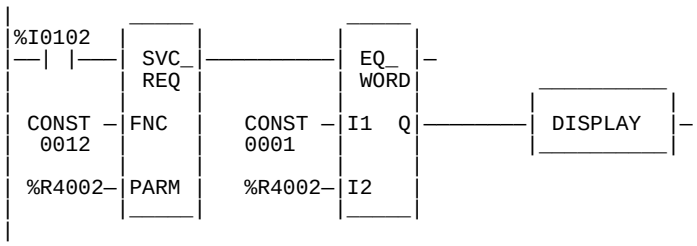
Of the CPUs discussed in this manual, Service Request 12 is supported *only* by 90-30 CPUs, beginning with Release 8.0.

The parameter block is an output parameter block only; it has a length of one word.

1 = run/disabled	address
2 = run/enabled	

Example:

In the following example, the PLC run state is always read into location %R4002. If the state is Run/Disabled, the CALL function calls program block DISPLAY.



SVCREQ #13: Shut Down (Stop) PLC

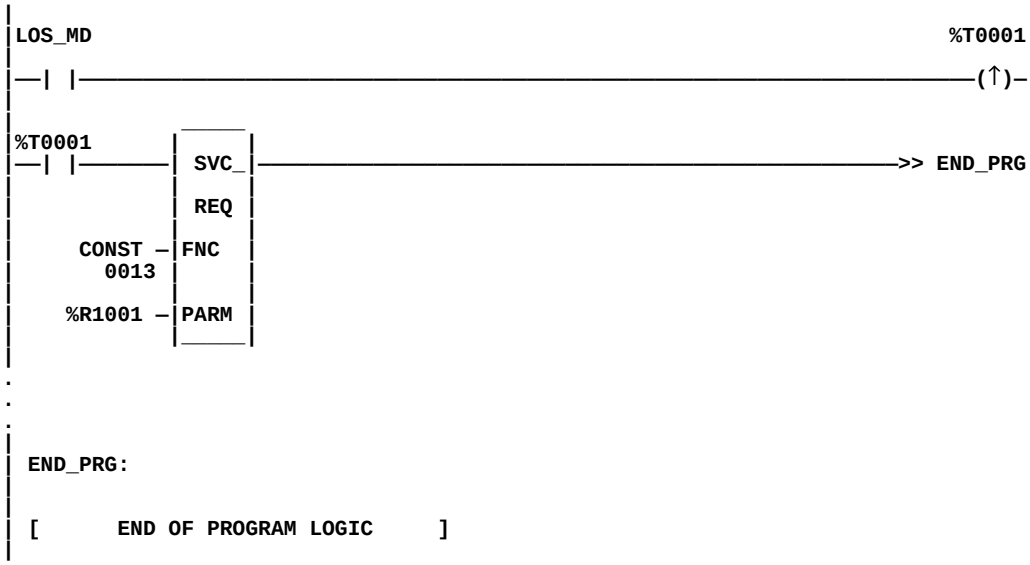
Use SVCREQ function #13 in order to stop the PLC ***at the end of the next sweep***. All outputs will go to their designated default states at the beginning of the next PLC sweep. An informational fault is placed in the PLC fault table, noting that a "SHUT DOWN PLC" function block was executed. The I/O scan will continue as configured.

This function has no parameter block.

Example:

In the following example, when a "Loss of I/O Module" fault occurs, SVCREQ function #13 executes. Since no parameter block is needed, the PARM input is not used; however, the programming software requires that an entry be made for PARM.

This example uses a JUMP to the end of the program to force a shutdown if the Shutdown PLC function executes successfully. This JUMP and LABEL are needed because the transition to **STOP** mode does not occur until the end of the sweep in which the function executes.



Note

To ensure that the %S0002 LST_SCN contact will operate correctly, the PLC will execute one additional sweep after the sweep in which the SVCREQ function #13 was executed.

SVCREQ #14: Clear Fault Tables

Use SVCREQ function #14 in order to clear either the PLC fault table or the I/O fault table. The SVCREQ output is set ON unless some number other than 0 or 1 is entered as the requested operation (see below).

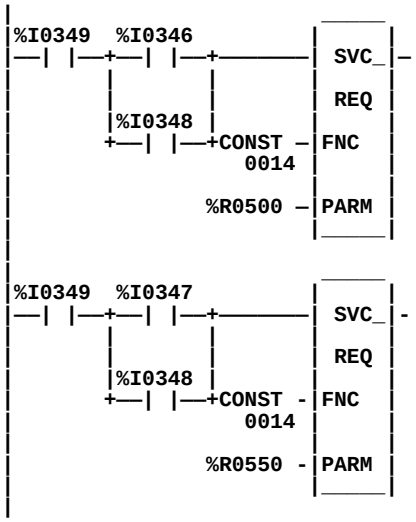
For this function, the parameter block has a length of 1 word. It is an input parameter block only.

0 = clear PLC fault table. 1 = clear I/O fault table.	address

Example:

In the following example, when input %I0346 is on and input %I0349 is on, the PLC fault table is cleared. When input %I0347 is on and input %I0349 is on, the I/O fault table is cleared. When input %I0348 is on and input %I0349 is on, both are cleared.

The parameter block for the PLC fault table is located at %R0500; for the I/O fault table the parameter block is located at %R0550. Both parameter blocks are set up elsewhere in the program.



SVCREQ #15: Read Last-Logged Fault Table Entry

Use SVCREQ function #15 in order to read the last entry logged in either the PLC fault table or the I/O fault table. The SVCREQ output is set ON unless some number other than 0 or 1 is entered as the requested operation (see below), or the fault table is empty. (For additional information on fault table entries, refer to chapter 3, “Fault Explanations and Correction.”)

For this function, the parameter block has a length of 22 words. The input parameter block has this format:

0 = Read PLC fault table. 1 = Read I/O fault table.	address

The format for the output parameter block depends on whether the function reads data from the PLC fault table or the I/O fault table.

PLC Fault Table Output Format

Low Byte	High Byte		
0			
long/short		address + 1	
spare		address + 2	
PLC fault address		address + 3	
		address + 4	
fault group and action		address + 5	
error code		address + 6	
		address + 7	
		address + 8	
		address + 9	
		address + 10	
		address + 11	
fault specific data		address + 12	
		address + 13	
		address + 14	
		address + 15	
		address + 16	
		address + 17	
		address + 18	
time stamp		address + 19	
		address + 20	
		address + 21	

I/O Fault Table Output Format

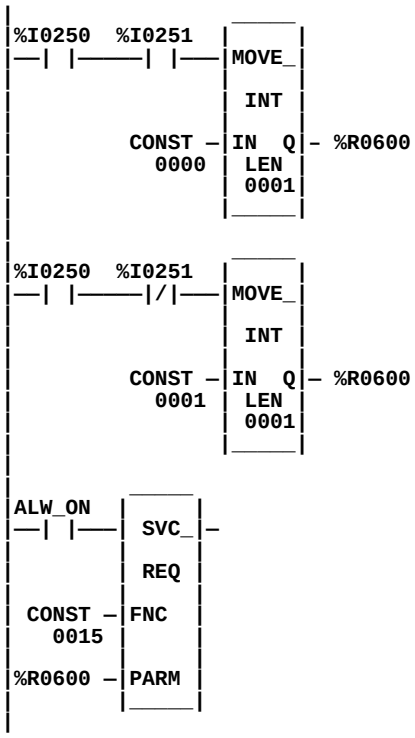
Low Byte	High Byte
1	
long/short	
reference address	
I/O fault address	
fault group and action	
fault category	fault type
fault description	
fault specific data	
time stamp	

In the first byte of word address + 1, the Long/Short indicator defines the quantity of fault specific data present in the fault entry. It may be:

```
PLC Fault Table:   00 = -8 bytes (short)
                   01 = 24 bytes (long)
I/O Fault Table:   02 = –5 bytes (short)
                   03 = 21 bytes (long)
```

Example 1:

In the following example, when input %I0251 is on and input %I0250 is on, the last entry in the PLC fault table is read into the parameter block. When input %I0251 is off and input %I0250 is on, the last entry in the I/O fault table is read into the parameter block. The parameter block is located at location %R0600.



Example 2:

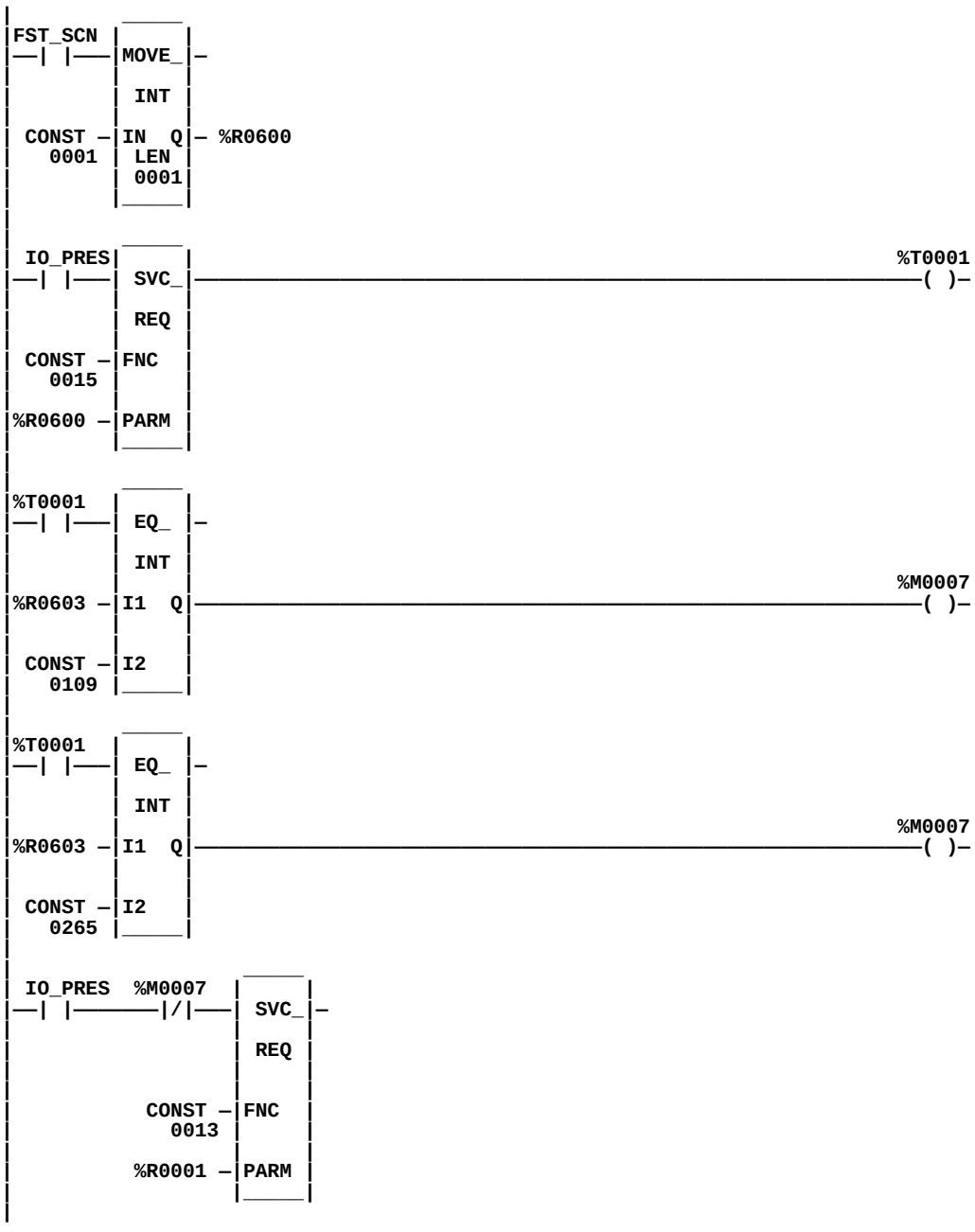
In the next example, the PLC is shut down when any fault occurs on an I/O module except when the fault occurs on modules in rack 0, slot 9 and in rack 1, slot 9. If faults occur on these two modules, the system remains running. The parameter for "table type" is set up on the first sweep. The contact IO_PRES, when set, indicates that the I/O fault table contains an entry. The PLC CPU sets the normally open contact in the sweep after the fault logic places a fault in the table. If faults are placed in the table in two consecutive sweeps, the normally open contact is set for two consecutive sweeps.

The example uses a parameter block located at %R0600. After the SVCREQ function executes, the fourth, fifth, and sixth words of the parameter block contain the address of the I/O module that faulted:

1		%R0600
long/short		%R0601
reference address		%R0602
rack number	slot number	%R0603
I/O bus no.	bus address	%R0604
point address		%R0605

fault data

In the program, the EQ_INT blocks compare the rack/slot address in the table to hexadecimal constants. The internal coil %M0007 is turned on when the rack/slot where the fault occurred meets the criteria specified above. If %M0007 is on, its normally closed contact is off, preventing the shutdown. Conversely, if %M0007 is off because the fault occurred on a different module, the normally closed contact is on and the shutdown occurs.


FST_SCN
MOVE
INT
CONST 0001
IN
LEN
0001
Q
%R0600
IO_PRES
SVC_REQ
FNC
CONST 0015
PARM
%R0600
%T0001
%T0001
EQ
INT
I1
I2
CONST 0109
%R0603
%M0007
%T0001
EQ
INT
I1
I2
CONST 0265
%R0603
%M0007
IO_PRES
%M0007
SVC_REQ
FNC
CONST 0013
PARM
%R0001

SVCREQ #16: Read Elapsed Time Clock

Use the SVCREQ function with function number 16 in order to read the value of the system's elapsed time clock. This clock tracks elapsed time in seconds since the PLC powered on. The timer will roll over approximately once every 100 years.

This function has an output parameter block only. The parameter block has a length of 3 words.

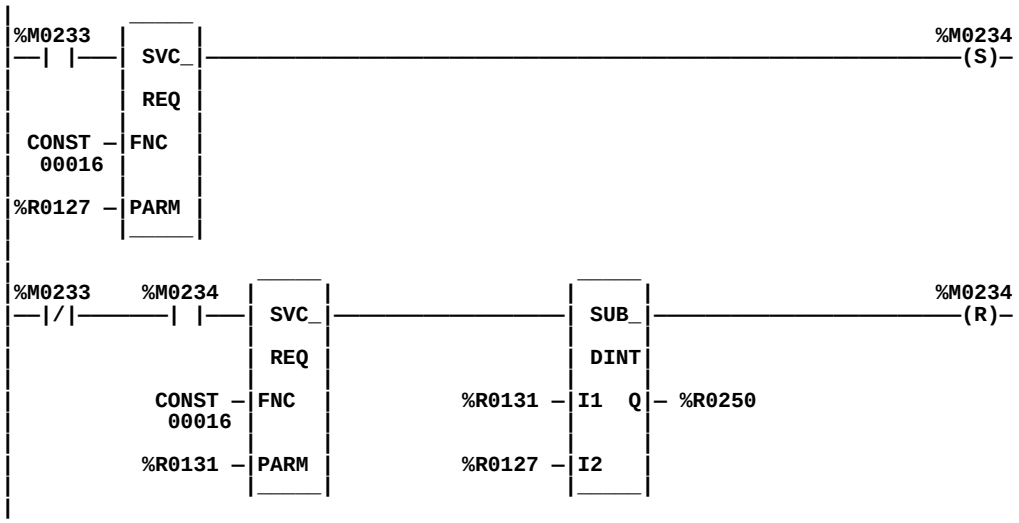
seconds from power on (low order)	address
seconds from power on (high order)	address + 1
100 microsecond ticks	address + 2

The first two words are the elapsed time in seconds. The last word is the number of 100 microsecond ticks in the current second.

Example:

In the following example, when internal coil %M0233 is on, the value of the elapsed time clock is read and coil %M0234 is set. When it is off, the value is read again. The difference between the values is then calculated, and the result is stored in register memory at location %R0250.

The parameter block for the first read is at %R0127; for the second read, at %R0131. The calculation ignores the number of hundred microsecond ticks and the fact that the DINT type is actually a signed value. The calculation is correct until the time since power-on reaches approximately 50 years.



SVCREQ #18: Read I/O Override Status

Use SVCREQ function #18 in order to read the current status of overrides in the CPU.

Note

This feature is available ***only*** for 331 or higher CPUs.

For this function, the parameter block has a length of 1 word. It is an output parameter block only.

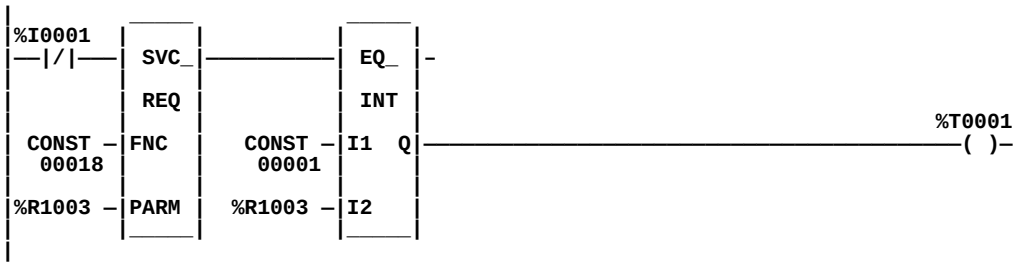
0 = No overrides are set. 1 = Overrides are set.	address

Note

SVCREQ #18 reports only overrides of %I and %Q references.

Example:

In the following example, the status of I/O overrides is always read into location %R1003. If any overrides are present, output %T0001 is set on.



SVCREQ #23: Read Master Checksum

Use SVCREQ function #23 to read the master checksums for the user program and the configuration. The SVCREQ output is always set to ON if the function is enabled, and the output block of information (see below) starts at the address given in parameter 3 (PARM) of the SVCREQ function.

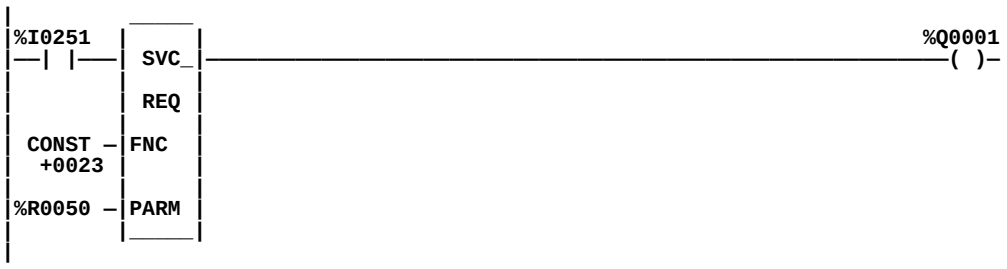
When a **RUN MODE STORE** is active, the program checksums may not be valid until the store is complete. Therefore, two flags are provided at the beginning of the output parameter block to indicate when the program and configuration checksums are valid.

For this function, the output parameter block has a length of 12 words with this format:

Master Program Checksum Valid (0 = not valid, 1 = valid)	address
Master Configuration Checksum Valid (0 = not valid, 1 = valid)	address + 1
Number of Program Blocks (including _MAIN)	address + 2
Size of User Program in Bytes (DWORD data type)	address + 3
Program Additive Checksum	address + 5
Program CRC Checksum (DWORD data type)	address + 6
Size of Configuration Data in Bytes	address + 8
Configuration Additive Checksum	address + 9
Configuration CRC Checksum (DWORD data type)	address + 10

Example:

In the following example, when input %I0251 is ON, the master checksum information is placed into the parameter block, and the output coil (%Q0001) is turned on. The parameter block is located at %R0050.



SVCREQ #26/30: Interrogate I/O

Use SVCREQ function #26 (or #30—they are identical; i.e., you can use either number to accomplish the same thing) to interrogate the actual modules present and compare them with the rack/slot configuration, generating addition, loss, and mismatch alarms, as if a store configuration had been performed. This SVCREQ will generate faults on both the PLC and I/O fault tables, depending on the fault.

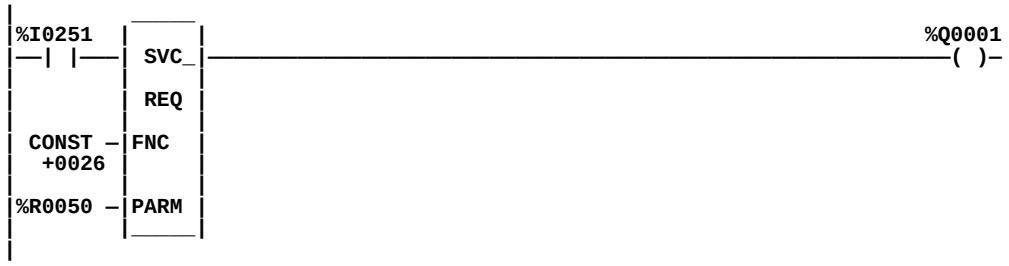
This function has no parameter block and always outputs power flow.

Note

The time for this SVCREQ to execute depends on how many faults exist. Therefore, execution time of this SVCREQ will be greater for situations where more modules are at fault.

Example:

In the following example, when input %I0251 is ON, the actual modules are interrogated and compared to the rack/slot configuration. Output %Q0001 is turned on after the SVCREQ is complete.



Note

This Service Request is not available on Micro PLCs.

SVCREQ #29: Read Elapsed Power Down Time

Use the SVCREQ function #29 to read the the amount of time elapsed between the last power-down and the most recent power-up. The SVCREQ output is always set to ON, and the output block of information (see below) starts at the address given in parameter 3 (PARM) of the SVCREQ function.

Note

This function is available only in the 331 or higher CPUs.

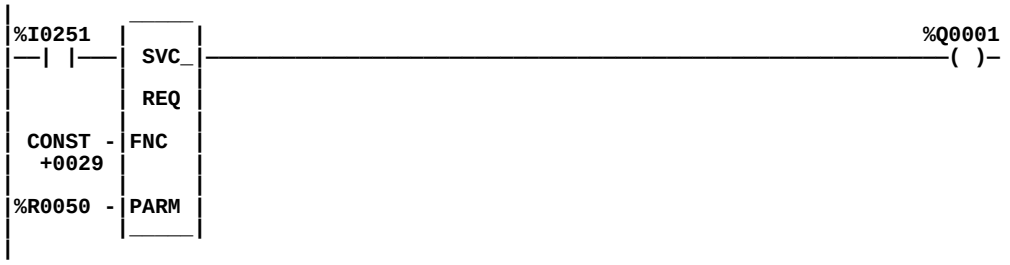
This function has an output parameter block only. The parameter block has a length of 3 words.

Power-Down Elapsed Seconds (low order)	address
Power-Down Elapsed Seconds (high order)	address + 1
100 Microsecond ticks	address + 2

The first two words are the power-down elapsed time in seconds. The last word is the remaining power-down elapsed time in 100 microsecond ticks (which is always 0). Whenever the PLC can not properly calculate the power down elapsed time, the time will be set to 0. This will happen when the PLC is powered up with CLR M/T pressed on the HHP. This will also happen if the watchdog timer times out before power-down.

Example:

In the following example, when input %I0251 is ON, the Elapsed Power-Down Time is placed into the parameter block, and the output coil (%Q0001) is turned on. The parameter block is located at %R0050.



SVCREQ #46:Fast Backplane Status Access

Use the SVCREQ function #46 to perform one of the following fast backplane access functions:

1. Read a word of extra status data from one of more specified smart modules.
2. Write a word of extra status data from one of more specified smart modules.
3. Read/Write: Read a word of extra status data from one or more specified modules and write the data value between 0 and 15 to the same module, all in one operations.

Note

This Service Request is available only for use with modules that support it. Currently, the only module designed to support this function is the DSM (Digital Servo Module) 312 Version. This DSM module is not available at the time of publication of this manual; however, it is scheduled for release soon.

This Service Request has a variable length as described below. The first word of the parameter block (PARM) has this format:

1 = Read extra data 2= Write extra data 3 = Read/write extra data	address

The first word of the parameter block determines which function will be used.

Read Extra Status Data (Function #1)

The Read Extra Data function reads a word of extra status data from each of the modules specified by a list in the parameter block and places the status data values into the parameter block. The parameter block requires (N + 4) words of reference memory, where N is the number of modules to which the data will be written.

Use the table on the following page to interpret the output values:

Location	Field	Meaning
Address	Function	1 = read extra status data
Address + 1	Error Code	An error code is placed here if the function fails because any of the modules is not present, inappropriate, or not working.
Address + 2	First rack & slot	Rack and slot number (in the form RRSS in hexadecimal, where RR is the rack number and SS is the slot number) of the first module from which the data will be read
Address + 3	Read data from first module	The data read from the first module will be place here
Address + 4	Second rack & slot	Rack and slot number (in the form RRSS in hexadecimal, where RR is the rack number and SS is the slot number) of the second module from which the data will be read
Address + 5	Read data from second module	The data read from the second module will be place here
Address + (I * 2)	I^{th} rack & slot	Rack and slot number (in the form RRSS in hexadecimal, where RR is the rack number and SS is the slot number) of the I^{th} module from which the data will be read
Address + (I * 2) + 1	Read data from I^{th} module	The data read from the I^{th} module will be place here
Address + (N * 2)	Last rack & slot	Rack and slot number (in the form RRSS in hexadecimal, where RR is the rack number and SS is the slot number) of the last module from which the data will be read
Address + (N * 2) + 1	Read data from last module	The data read from the last module will be place here
Address + (N * 2) + 2	End of list indicator	A zero in this word indicates the end of the list of modules

Write Data (Function #2)

The write data function writes a data value between 0 and 15 from the parameter block to one or more modules specified by a list in the parameter block. The parameter block requires (N + 4) words of reference memory, where N is the number of modules to which the data will be written.

Location	Field	Meaning
Address	Function	2 = write data
Address + 1	Error Code	An error code is placed here if the function fails because any of the modules is not present, inappropriate, or not working. No error code is set if the function executes but any of the modules does not receive the write data properly
Address + 2	First rack & slot	Rack and slot number (in the form RRSS in hexadecimal, where RR is the rack number and SS is the slot number) of the first module to which the data will be sent
Address + 3	Write data for first module	This data value will be written to the first module
Address + 4	Second rack & slot	Rack and slot number (in the form RRSS in hexadecimal, where RR is the rack number and SS is the slot number) of the second module to which the data will be sent
Address + 5	Write data for second module	This data value will be written to the second module
Address + (I * 2)	I^{th} rack & slot	Rack and slot number (in the form RRSS in hexadecimal, where RR is the rack number and SS is the slot number) of the I^{th} module to which the data will be sent
Address + (I * 2) + 1	Write data for I^{th} module	This data value will be written to the I^{th} module
Address + (N * 2)	Last rack & slot	Rack and slot number (in the form RRSS in hexadecimal, where RR is the rack number and SS is the slot number) of the last module to which the data will be sent
Address + (N * 2) + 1	Write data for last module	This data value will be written to the last module
Address + (N * 2) + 2	End of list indicator	A zero in this word indicates the end of the list of modules

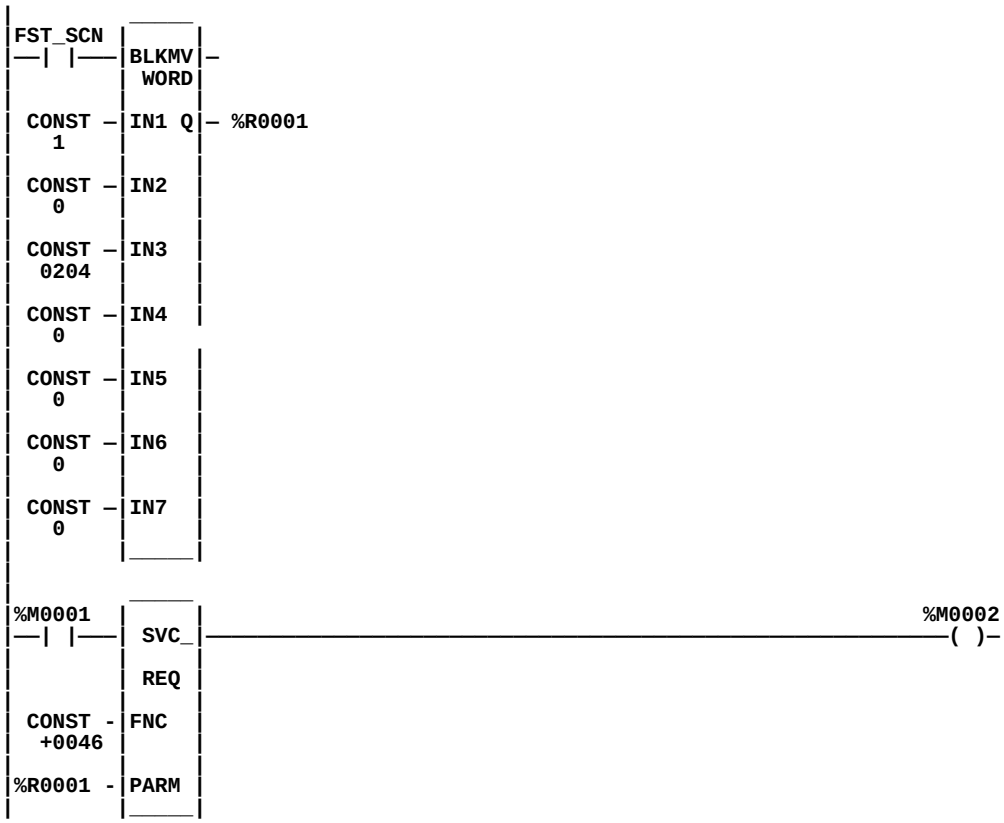
Read/Write Data (Function #3)

The read/write function reads a word of extra status data from a module specified in the parameter block, then writes a data value between 0 and 15 from the parameter block to that module. This read write process is repeated for each module in a list in the parameter block. The parameter block (N * 3) + 3 words of reference memory, where N is the number of modules with which data will be exchanged.

Location	Field	Meaning
Address	Function	3 = read/write
Address + 1	Error Code	An error code is placed here if the function fails because any of the modules is not present, inappropriate, or not working. No error code is set if the function executes but any of the modules does not receive the write data properly
Address + 2	First rack & slot	Rack and slot number (in the form RRSS in hexadecimal, where RR is the rack number and SS is the slot number) of the first module with which data will be exchanged
Address + 3	Read data from first module	The data read from the first module will be placed here
Address + 4	Write data for first module	This data value will be written to the first module
Address + 5	Second rack & slot	Rack and slot number (in the form RRSS in hexadecimal, where RR is the rack number and SS is the slot number) of the second module with which data will be exchanged
Address + 6	Read data from second module	The data read from the second module will be placed here
Address + 7	Write data for second module	This data value will be written to the second module
Address + ((I-1) * 3) + 2	I^{th} rack & slot	Rack and slot number (in the form RRSS in hexadecimal, where RR is the rack number and SS is the slot number) of the I^{th} module with which data will be exchanged
Address + ((I-1) * 3) + 3	Read data from Ith module	The data read from the I^{th} module will be placed here
Address + ((I-1) * 3) + 4	Write data for Ith module	This data value will be written to the I^{th} module
Address + ((N-1) * 3) + 2	Last rack & slot	Rack and slot number (in the form RRSS in hexadecimal, where RR is the rack number and SS is the slot number) of the last module with which data will be exchanged
Address + ((N-1) * 3) + 3	Read data from last module	The data read from the last module will be placed here
Address + ((N-1) * 3) + 4	Write data for last module	This data value will be written to the last module
Address + (N * 3) + 2	End of list indicator	A zero in this word indicates the end of the list of modules

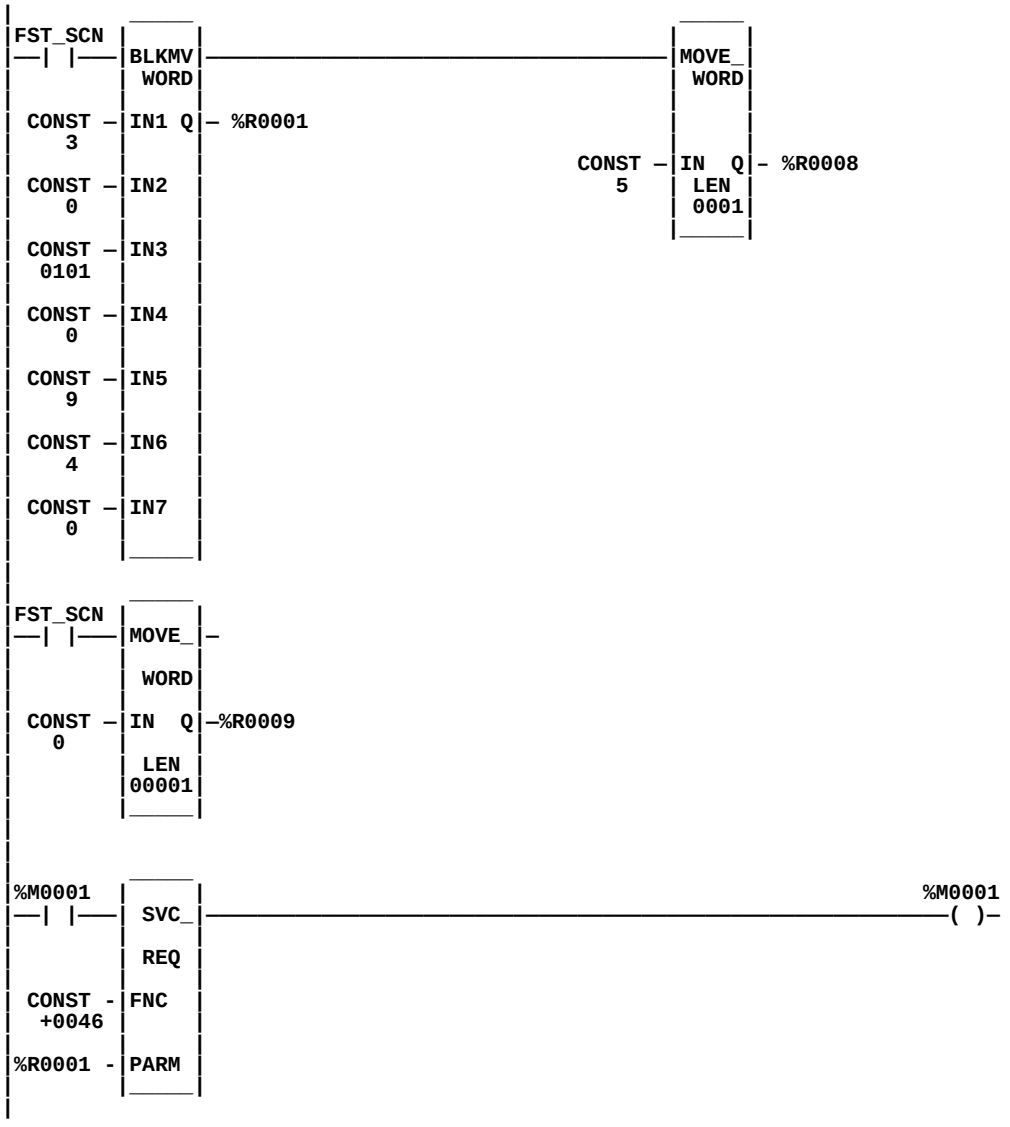
Example 1:

The following example shows a Read of a single module at Rack 2, Slot 4. IN4 and IN5 must be set to zero (0). IN6 and IN7 are not important in this example. If the function completes successfully, the data will be in %R0004.



Example 2:

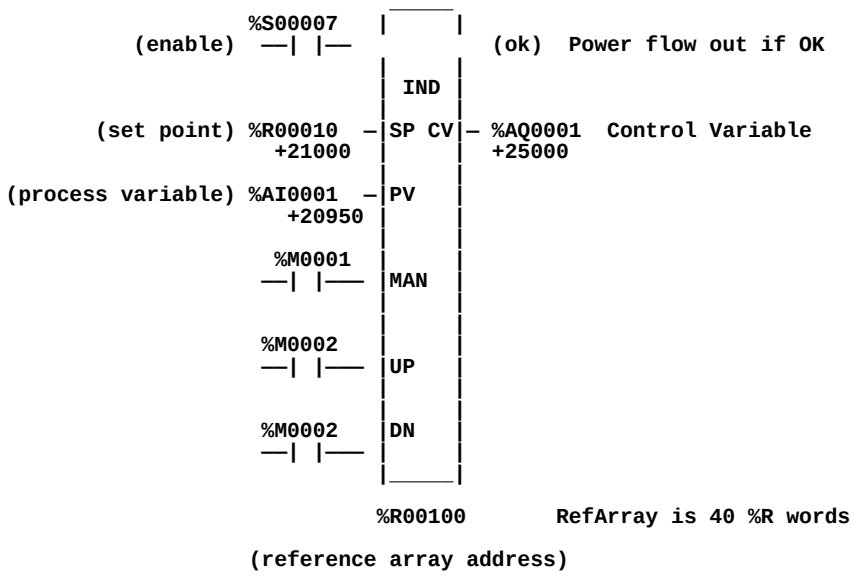
This example reads the extra status data from the module in Rack 0, Slot 4 and from the module in Rack 1, Slot 1. It writes a 5 to the first module and a 9 to the second. Note that the modules do not need to be listed in order by slot numbers. Data read from the module in Rack 0, Slot 4 will be placed into %R0007. Data read from the module in Rack 1, Slot 1 will be placed in %R0004.



PID

The Proportional plus Integral plus Derivative (PID) control function is the best known general purpose algorithm for closed loop process control. The Series 90 PID function block compares a Process Variable feedback with a desired process Set Point and updates a Control Variable output based on the error.

The block uses PID loop gains and other parameters stored in an array of 40 16 bit words (discussed on page 4-173) to solve the PID algorithm at the desired time interval. All parameters are 16 bit integer words for compatibility with 16 bit analog process variables. This allows %AI memory to be used for input Process Variables and %AQ to be used for output Control Variables. The example shown below includes typical inputs.



As the input Set Point and Process Variable and output Control Variable terms are used so frequently, they will be abbreviated as SP, PV and CV. As scaled 16 integer numbers, many parameters must be defined in either PV counts or units or CV counts or units. For example, the SP input must be scaled over the same range as PV as the PID block calculates the error by subtracting these two inputs. The PV and CV Counts may be –32000 or 0 to 32000 matching analog scaling or from 0 to 10000 to display variables as 0.00% to 100.00%. The PV and CV Counts do not have to have the same scaling, in which case there will be scale factors included in the PID gains.

Note

The PID will not execute more often than once every 10 milliseconds. This could change your desired results if you set it up to execute every sweep and the sweep is under 10 milliseconds. In such a case, the PID function will not run until enough sweeps have occurred to accumulate an elapsed time of 10 milliseconds; e.g., if the sweep time is 9 milliseconds, the PID function will execute every other sweep with an elapsed time of 18 milliseconds for every time it executes.

Parameters:

Parameter	Description
enable	When enabled through a contact, the PID function is performed.
SP	SP is the control loop or process set point. Set using PV Counts, the PID adjusts the output CV so that PV matches SP (zero error).
PV	Process Variable input from the process being controlled, often a %AI input.
MAN	When energized to 1 (through a contact), the PID block is in **MANUAL** mode. If the PID block is on manual off, the PID block is in automatic mode.
UP	If energized along with MAN, it adjusts the CV up by 1 CV per solution.*
DN	If energized along with MAN, it adjusts the CV down by 1 CV per solution.*
RefArray Address	Address is the location of the PID control block information (user and internal parameters). Uses 40 %R words that cannot be shared.
ok	The ok output is energized when the function is performed without error. It is off if error(s) exist.
CV	CV is the control variable output to the process, often a %AQ analog output.

*****Incremented (UP parameter) or decremented (DN parameter) by one (1) per access of the PID function.

Valid Memory Types:

Parameter	flow	%I	%Q	%M	%T	%S	%G	%R	%AI	%AQ	const	none
enable	•											
SP		•	•	•	•		•	•	•	•	•	
PV		•	•	•	•		•	•	•	•		
MAN	•											
UP	•											
DN	•											
address								•				
ok	•											•
CV		•	•	•	•		•	•	•	•		

• Valid reference or place where power may flow through the function.

PID Parameter Block:

Besides the 2 input words and the 3 Manual control contacts, the PID block uses 13 of the parameters in the RefArray. These parameters must be set before calling the block. The other parameters are used by the PLC and are non-configurable. The %Ref shown in the table below is the same RefArray Address at the bottom of the PID block. The number after the plus sign is the offset in the array. For example, if the RefArray starts at %R100, the %R113 will contain the Manual Command used to set the Control Variable and the integrator in Manual mode.

Table 4-4. PID Parameters Overview

Register	Parameter	Low Bit Units	Range of Values
%Ref+0000	Loop Number	Integer	0 to 255 (for user display only)
%Ref+0001	Algorithm	N/A; set and maintained by the PLC	Non-configurable
%Ref+0002	Sample Period	10 milliseconds	0 (every sweep) to 65535 (10.9 Min). Use at least 10 for 90-30 PLCs (see Note on page 4-171).
%Ref+0003	Dead Band +	PV Counts	0 to 32000 (never negative)
%Ref+0004	Dead Band —	PV Counts	–32000 to 0 (never positive)
%Ref+0005	Proportional Gain –Kp	0.01 CV%/PV%	0 to 327.67 %/%
%Ref+0006	Derivative Gain–Kd	0.01 seconds	0 to 327.67 sec
%Ref+0007	Integral Rate–Ki	Repeat/1000 Sec	0 to 32.767 repeat/sec
%Ref+0008	CV Bias/Output Offset	CV Counts	–32000 to 32000 (add to integrator output)
%Ref+0009	Upper Clamp	CV Counts	–32000 to 32000(>%Ref+10) output limit
%Ref+0010	Lower Clamp	CV Counts	–32000 to 32000(<%Ref+09) output limit
%Ref+0011	Minimum Slew Time	Second/Full Travel	0 (none) to 32000 sec to move 32000 CV
%Ref+0012	Config Word	Low 5 bits used	Bit 0 to 2 for Error+/–, OutPolarity, Deriv.
%Ref+0013	Manual Command	CV Counts	Tracks CV in Auto or Sets CV in Manual
%Ref+0014	Control Word	Maintained by the PLC, ***unless*** Bit 1 is set.	PLC maintained unless set otherwise: low bit sets Override if 1 (see description in the “PID Parameters Details” table on page 4-174)
%Ref+0015	Internal SP	N/A; set and maintained by the PLC	Non-configurable
%Ref+0016	Internal CV	N/A; set and maintained by the PLC	Non-configurable
%Ref+0017	Internal PV	N/A; set and maintained by the PLC	Non-configurable
%Ref+0018	Output	N/A; set and maintained by the PLC	Non-configurable

Table 4-4. PID Parameters Overview (Continued)

Register	Parameter	Low Bit Units	Range of Values
%Ref+0019	Diff Term Storage	N/A; set and maintained by the PLC	Non-configurable
%Ref+0020 and %Ref+0021	Int Term Storage	N/A; set and maintained by the PLC	Non-configurable
%Ref+0022	Slew Term Storage	N/A; set and maintained by the PLC	Non-configurable
%Ref+0023	Clock	N/A; set and	Non-configurable
%Ref+0024		maintained by	
%Ref+0025	(time last executed)	the PLC	
%Ref+0026	Y Remainder Storage	N/A; set and maintained by the PLC	Non-configurable
%Ref+0027	Lower Range for SP, PV	PV Counts	–32000 to 32000 (>%Ref+28) for display
%Ref+0028	Upper Range for SP, PV	PV Counts	–32000 to 32000 (<%Ref+27) for display
%Ref+0029 • %Ref+0034	Reserved for internal use	N/A	Non-configurable
%Ref+0035 • %Ref+0039	Reserved for external use	N/A	Non-configurable

The RefArray array must be %R registers on the 90-30 PLC. Note that every PID block call must use a different 40-word array even if all 13 user parameters are the same because other words in the array are used for internal PID data storage. Make sure the array does not extend beyond the end of memory.

To configure the user parameters, select the PID function and press **F10** to zoom in to a screen displaying User Parameters; then use arrow keys to select fields and type in desired values. You can use 0 for most default values, except the CV Upper Clamp, which must be greater than the CV Lower Clamp for the PID block to operate. Note that the PID block does **not** pass power if there is an error in User Parameters, so monitor with a temporary coil while modifying data.

Once suitable PID values have been chosen, they should be defined as constants in the BLKMOV so that they can be used to reload default PID user parameters if needed.

Operation of the PID Instruction

Normal Automatic operation is to call the PID block every sweep with power flow to Enable and no power flow to Manual input contacts. The block compares the current PLC elapsed time clock with the last PID solution time stored in the internal RefArray. If the time difference is greater than the sample period defined in the third word (%Ref+2) of the RefArray, the PID algorithm is solved using the time difference and both the last solution time and Control Variable output are updated. In Automatic mode, the output Control Variable is placed in the Manual Command parameter %Ref+13.

If power flow is provided to both Enable and Manual input contacts, the PID block is placed in Manual mode and the output Control Variable is set from the Manual Command parameter %Ref+13. If either the UP or DN inputs have power flow, the Manual Command word is incremented or decremented by one CV count every PID solution. For faster manual changes of the output Control Variable, it is also possible to add or subtract any CV count value directly to/from the Manual Command word.

The PID block uses the CV Upper and CV Lower Clamp parameters to limit the CV output. If a positive Minimum Slew Time is defined, it is used to limit the rate of change of the CV output. If either the CV amplitude or rate limit is exceeded, the value stored in the integrator is adjusted so that CV is at the limit. This anti-reset windup feature (defined on page 4-178) means that even if the error tried to drive CV above (or below) the clamps for a long period of time, the CV output will move off the clamp as soon as the error term changes sign.

This operation, with the Manual Command tracking CV in Automatic mode and setting CV in Manual mode, provides a bumpless transfer between Automatic and Manual modes. The CV Upper and Lower Clamps and the Minimum Slew Time still apply to the CV output in Manual mode and the internal value stored in the integrator is updated. This means that if you were to step the Manual Command in Manual mode, the CV output will not change any faster that the Minimum Slew Time (Inverse) rate limit and will not go above or below the CV Upper or CV Lower Clamp limits.

Note

A specific PID function should not be called more than once per sweep.

The following table provides more details about the parameters discussed briefly in Table 4-4. The number in parentheses after each parameter name is the offset in the RefArray.

Table 4-5. PID Parameters Details

Data Item	Description
Loop Number (00)	This is an optional parameter available to identify a PID block. It is an unsigned integer that provides a common identification in the PLC with the loop number defined by an operator interface device. The loop number is displayed under the block address when logic is monitored from the Logicmaster 90-30/20/Micro software.
Algorithm (01)	An unsigned integer that is set by the PLC to identify what algorithm is being used by the function block. The ISA algorithm is defined as algorithm 1, and the independent algorithm is identified as algorithm 2.
Sample Period (02)	The shortest time, in 10 millisecond increments, between solutions of the PID algorithm. For example, use a 10 for a 100 millisecond sample period. If it is 0, the algorithm is solved every time the block is called (see section below on PID block scheduling). The PID algorithm is solved only if the current PLC elapsed time clock is at or later than the last PID solution time plus this Sample Period. Remember, that the 90-30 will not use a solution time less than 10 milliseconds (see Note on page 4-171); so sweeps will be skipped for smaller sweep times. This function compensates for the actual time elapsed since the last execution, within 100 microseconds. If this value is set to 0, the function is executed each time it is enabled; however, it is restricted to a minimum of 10 milliseconds as noted above.
Dead Band (+/—) (03/04)	INT values defining the upper (+) and lower (–) Dead Band limits in PV Counts. If no Dead Band is required, these values must be 0. If the PID Error (SP – PV) or (PV – SP) is above the (–) value and below the (+) value, the PID calculations are solved with an Error of 0. If non-zero, the (+) value must be greater than 0 and the (–) value less than 0 or the PID block will not function. *You should leave these at 0 until the PID loop gains are setup or tuned*. After that, you may want to add Dead Band to avoid small CV output changes due to small variations in error, perhaps to reduce mechanical wear.
Proportional Gain–Kp (05)	This INT number, called the Controller gain, Kc, in the ISA version, determines the change in CV in CV Counts for a 100 PV Count change in the Error term. It is displayed as 0.00 %/% with an implied decimal point of 2 . For example, a Kp entered as 450 will be displayed as 4.50 and will result in a Kp*Error/100 or 450*Error/100 contribution to the PID Output. Kp is generally the first gain set when adjusting a PID loop.
Derivative Gain–Kd (06)	This INT number determines the change in CV in CV Counts if the Error or PV changes 1 PV Count every 10 milliseconds. Entered as a time with the low bit indicating 10 milliseconds, it is displayed as 0.00 Seconds with an implied decimal point of 2. For example, a Kd entered as 120 will be displayed as 1.20 Sec and will result in a Kd * delta Error/delta time or 120*4/3 contribution to the PID Output if Error was changing by 4 PV Counts every 30 milliseconds. Kd can be used to speed up a slow loop response, but is very sensitive to PV input noise.
Integral Rate Gain–Ki (07)	This INT number determines the change in CV in CV Counts if the Error were a constant 1 PV Count. It is displayed as 0.000 Repeats/Sec with an implied decimal point of 3. For example, a Ki entered as 1400 will be displayed as 1.400 Repeats/Sec and will result in a Ki * Error *dt or 1400 * 20 * 50/1000 contribution to PID Output for an Error of 20 PV Counts and a 50 millisecond PLC sweep time (Sample Period of 0). Ki is usually the second gain set after Kp.
CV Bias/Output Offset (08)	An INT value in CV Counts added to the PID Output before the rate and amplitude clamps. It can be used to set non-zero CV values if only Kp Proportional gains are used, or for feed forward control of this PID loop output from another control loop.

Table 4-5. PID Parameters Details - Continued

Data Item	Description
CV Upper and Lower Clamps (09/10)	INT values in CV Counts that define the highest and lowest value for CV. These values are required and the Upper Clamp must have a more positive value than the Lower Clamp, or the PID block will not work. These are usually used to define limits based on physical limits for a CV output. They are also used to scale the Bar Graph display for CV for the LM90 or ADS PID display. The block has anti-reset windup to modify the integrator value when a CV clamp is reached.
Minimum Slew Time (11)	A positive value to define the minimum number of seconds for the CV output to move from 0 to full travel of 100% or 32000 CV Counts. It is an inverse rate limit on how fast the CV output can be changed. If positive, CV can not change more than 32000 CV Counts times Delta Time (seconds) divided by Minimum Slew Time. For example, if the Sample Period was 2.5 seconds and the Minimum Slew Time is 500 seconds, CV can not change more than 32000*2.5/500 or 160 CV Counts per PID solution. As with the CV Clamps, there is an anti-windup feature that adjusts the integrator value if the CV rate limit is exceeded. If Minimum Slew Time is 0, there is no CV rate limit. Make sure you set Minimum Slew Time to 0 while you are tuning or adjusting PID loop gains.
Config Word	The low 5 bits of this word are used to modify three standard PID settings. The other bits should be set to 0. Set the low bit to 1 to modify the standard PID Error Term from the normal (SP – PV) to (PV – SP), reversing the sign of the feedback term. This is for Reverse Acting controls where the CV must go down when the PV goes up. Set the second bit to a 1 to invert the Output Polarity so that CV is the negative of the PID output rather than the normal positive value. Set the fourth bit to 1 to modify the Derivative Action from using the normal change in the Error term to the change in the PV feedback term. The low 5 bits in the Config Word are defined in detail below: **Bit 0** = Error Term. When this bit is set to 0, the error term is SP — PV. When this bit is set to 1, the error term is PV — SP. **Bit 1** = Output Polarity. When this bit is set to 0, the CV output represents the output of the PID calculation. When it is set to 1, the CV output represents the negative of the output of the PID calculation. **Bit 2** = Derivative action on PV. When this bit is set to 0, the derivative action is applied to the error term. When it is set to 1, the derivative action is applied to PV. All remaining bits should be zero. **Bit 3** = Deadband action. When the Deadband action bit is set to zero, then no deadband action is chosen. If the error is within the deadband limits, then the error is forced to be zero. Otherwise the error is not affected by the deadband limits. If the Deadband action bit is set to one, then deadband action is chosen. If the error is within the deadband limits, then the error is forced to be zero. If, however, the error is outside the deadband limits, then the error is reduced by the deadband limit (error = error – deadband limit). **Bit 4** = Anti-reset windup action. When this bit is set to zero, the anti-reset windup action uses a reset back calculation. When the output is clamped, this replaces the accumulated Y remainder value (defined on page 4-178) with whatever value is necessary to produce the clamped output exactly. When the bit is set to one, this replaces the accumulated Y term with the value of the Y term at the start of the calculation. In this way, the pre-clamp Y value is held as long as the output is clamped. **NOTE:** The anti-reset windup action bit is only available on release 6.50 or later 90-30 CPUs. Remember that the bits are set in powers of 2. For example, to set Config Word to 0 for default PID configuration, you would add 1 to change the Error Term from SP–PV to PV–SP, or add 2 to change the Output Polarity from CV = PID Output to CV = – PID Output, or add 4 to change Derivative Action from Error rate of change to PV rate of change, etc.

Table 4-5. PID Parameters Details - Continued

Data Item	Description
Manual Command (13)	This is an INT value set to the current CV output while the PID block is in Automatic mode. When the block is switched to **Manual** mode, this value is used to set the CV output and the internal value of the integrator within the Upper and Lower Clamp and Slew Time limits.
Control Word (14)	This is an internal parameter that is normally left at 0. If the Override low bit is set to 1, this word and other internal SP, PV and CV parameters must be used for remote operation of this PID block (see below). This allows remote operator interface devices, such as a computer, to take control away from the PLC program. Caution: if you do not want this to happen, make use the Control Word is set to 0. If the low bit is 0, the next 4 bits can be read to track the status of the PID input contacts as long as the PID Enable contact has power. A discrete data structure with the first five bit positions in the following format: **Bit: Word Value: Function: Status or External Action if Override bit set to 1:** 0 1 Override If 0, monitor block contacts below. If 1, set them externally. 1 2 Manual/ Auto If 1, block is in **Manual** mode; other numbers it is in **Automatic** mode. 2 4 Enable Should normally be 1; otherwise block is never called. 3 8 UP/Raise If 1 and Manual (Bit 1) is 1, CV is being incremented every solution. 4 16 DN/Lower If 1 and Manual (Bit 1) is 1, CV is being incremented every solution.
SP (15)	(Non-configurable–set and maintained by the PLC) Tracks SP in; must be set externally if Override = 1.
CV (16)	(Non-configurable–set and maintained by the PLC) Tracks CV out.
PV (17)	(Non-configurable–set and maintained by the PLC) Tracks PV in; must be set externally if Override bit = 1.
Output (18)	(Non-configurable–set and maintained by the PLC) This is a signed word value representing the output of the function block before the application of the optional inversion. If no output inversion is configured and the output polarity bit in the control word is set to 0, this value will equal the CV output. If inversion is selected and the output polarity bit is set to 1, this value will equal the negative of the CV output.
Diff Term Storage (19)	Used internally for storage of intermediate values. *Do not write to this location.*
Int Term Storage (20/21)	Used internally for storage of intermediate values. *Do not write to this location.*
Slew Term Storage (22)	Used internally for storage of intermediate values. *Do not write to this location.*
Clock (23–25)	Internal elapsed time storage (time last PID executed). *Do not write to these locations.*
Y Remainder (26)	Holds remainder for integrator division scaling for 0 steady state error.
Lower and Upper Range (27/28)	Optional INT values in PV Counts that define the highest and lowest display value for the SP and PV Logicmaster **Zoom** key horizontal bar graph and ADS PID faceplate display.
Reserved (29–34 and 35–39)	29–34 are reserved for internal use; 35–39 are reserved for external use. They are reserved for GE Fanuc use, and cannot be used for other purposes.

Internal Parameters in RefArray

As described in Table 4-6 on the previous pages, the PID block reads 13 user parameters and uses the rest of the 40 word RefArray for internal PID storage. Normally you would not need to change any of these values. If you are calling the PID block in Auto mode after a long delay, you may want to use SVC_REQ #16 to load the current PLC elapsed time clock into %Ref+23 to update the last PID solution time to avoid a step change on the integrator. If you have set the Override low bit of the Control Word (%Ref+14) to 1, the next four bits of the Control Word must be set to control the PID block input contacts (as described in Table 4-5 on the previous pages), and the Internal SP and PV must be set as you have taken control of the PID block away from the ladder logic.

PID Algorithm Selection (PIDISA or PIDIND) and Gains

The PID block can be programmed selecting either the Independent (PID_IND) term or standard ISA (PID_ISA) versions of the PID algorithm. The only difference in the algorithms is how the Integral and Derivative gains are defined. To understand the difference, you need to understand the following:

Both PID types calculate the Error term as SP – PV, which can be changed to Reverse Acting mode PV – SP if the Error Term (low bit 0 in the Config Word %Ref+12) is set to 1. Reverse Acting mode may be used if you want the CV output to move in the opposite direction from PV input changes (CV down for PV up) rather than the normal CV up for PV up.

Error = (SP – PV) or (PV – SP) if low bit of Config Word set to 1

The Derivative is normally based on the change of the Error term since the last PID solution, which may cause a large change in the output if the SP value is changed. If this is not desired, the third bit of the Config Word can be set to 1 to calculate the Derivative based on the change of the PV. The dt (or Delta Time) is determined by subtracting the last PID solution clock time for this block from the current PLC elapsed time clock.

dt = Current PLC Elapsed Time clock – PLC Elapsed Time Clock at Last PID solution

Derivative = (Error – previous Error)/dt or (PV – previous PV)/dt if 3rd bit of Config Word set to 1

The Independent term PID (PID_IND) algorithm calculates the output as:

PID Output = Kp * Error + Ki * Error * dt + Kd * Derivative + CV Bias

The standard ISA (PID_ISA) algorithm has a different form:

PID Output = Kc * (Error + Error * dt/Ti + Td * Derivative) + CV Bias

where Kc is the controller gain, and Ti is the Integral time and Td is the Derivative time. The advantage of ISA is that adjusting the Kc changes the contribution for the integral and derivative terms as well as the proportional one, which may make loop tuning easier. If you have PID gains in terms or Ti and Td, use

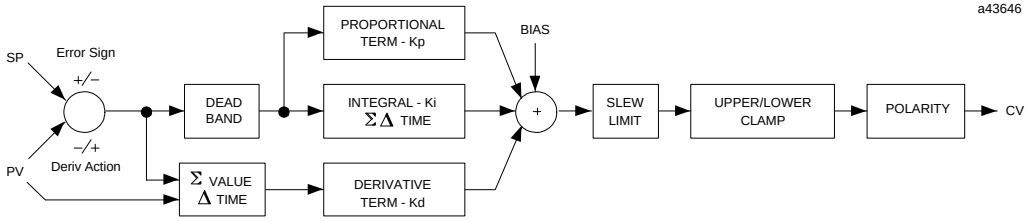
Kp = Kc Ki = Kc/Ti and Kd = Kc/Td

to convert them to use as PID User Parameter inputs.

The CV Bias term above is an additive term separate from the PID components. It may be required if you are using only Proportional Kp gain and you want the CV to be a non-zero value when the PV equals the SP and the Error is 0. In this case, set the CV Bias to the desired CV when the PV is at the SP. CV Bias can also be used for feed forward control where another PID loop or control algorithm is used to adjust the CV output of this PID loop.

If an Integral Ki gain is used, the CV Bias would normally be 0 as the integrator acts as an automatic bias. Just start up in Manual mode and use the Manual Command word (%Ref+13) to set the integrator to the desired CV, then switch to Automatic mode. This also works if Ki is 0, except the integrator will not be adjusted based on the Error after going into Automatic mode.

The following diagram shows how the PID algorithms work:



Independent Term Algorithm (PIDIND)

The ISA Algorithm (PIDISA) is similar except the Kp gain is factored out of Ki and Kd so that the integral gain is Kp * Ki and derivative gain is Kp * Kd. The Error sign, DerivAction and Polarity are set by bits in the Config Word user parameter.

CV Amplitude and Rate Limits

The block does not send the calculated PID Output directly to CV. Both PID algorithms can impose amplitude and rate of change limits on the output Control Variable. The maximum rate of change is determined by dividing the maximum 100% CV value (32000) by the Minimum Slew Time, if specified as greater than 0. For example, if the Minimum Slew Time is 100 seconds, the rate limit will be 320 CV counts per second. If the dt solution time was 50 milliseconds, the new CV output can not change more than 320*50/1000 or 16 CV counts from the previous CV output.

The CV output is then compared to the CV Upper and CV Lower Clamp values. If either limit is exceeded, the CV output is set to the clamped value. If either rate or amplitude limits are exceeded modifying CV, the internal integrator value is adjusted to match the limited value to avoid reset windup.

Finally, the block checks the Output Polarity (2nd bit of the Config Word %Ref+12) and changes the sign of the output if the bit is 1.

CV = Clamped PID Output or – Clamped PID Output if Output Polarity bit set

If the block is in Automatic mode, the final CV is placed in the Manual Command %Ref+13. If the block is in Manual mode, the PID equation is skipped as CV is set by the Manual Command, but all the rate and amplitude limits are still checked. That means that the Manual Command can not change the output above the CV Upper Clamp or below the CV Lower Clamps and the output can not change faster than the Minimum Slew Time allowed.

Sample Period and PID Block Scheduling

The PID block is a digital implementation of an analog control function, so the dt sample time in the PID Output equation is not the infinitesimally small sample time available with analog controls. The majority of processes being controlled can be approximated as a gain with a first or second order lag, possibly with a pure time delay. The PID block sets a CV output to the process and uses the process feedback PV to determine an Error to adjust the next CV output. A key process parameter is the total time constant, which is how fast does the PV respond when the CV is changed. As discussed in the Setting Loop Gains section below, the total time constant, Tp+Tc, for a first order system is the time required for PV to reach 63% of its final value when CV is stepped. The PID block will not be able to control a process unless its Sample Period is well under half the total time constant. Larger Sample Periods will make it unstable.

The Sample Period should be no bigger than the total time constant divided by 10 (or down to 5 worst case). For example, if PV seems to reach about 2/3 of its final value in 2 seconds, the Sample Period should be less than 0.2 seconds, or 0.4 seconds worst case. On the other hand, the Sample Period should not be too small, such as less than the total time constant divided by 1000, or the Ki * Error * dt term for the PID integrator will round down to 0. For example, a very slow process that takes 10 hours or 36000 seconds to reach the 63% level should have a Sample Period of 40 seconds or longer.

Unless the process is very fast, it is not usually necessary to use a Sample Period of 0 to solve the PID algorithm every PID sweep. If many PID loops are used with a Sample Period greater than the sweep time, there may be wide variations in PLC sweep time if many loops end up solving the algorithm at the same time. The simple solution is to sequence a one or more 1 bits through an array of bits set to 0 that is being used to enable power flow to individual PID blocks.

Determining the Process Characteristics

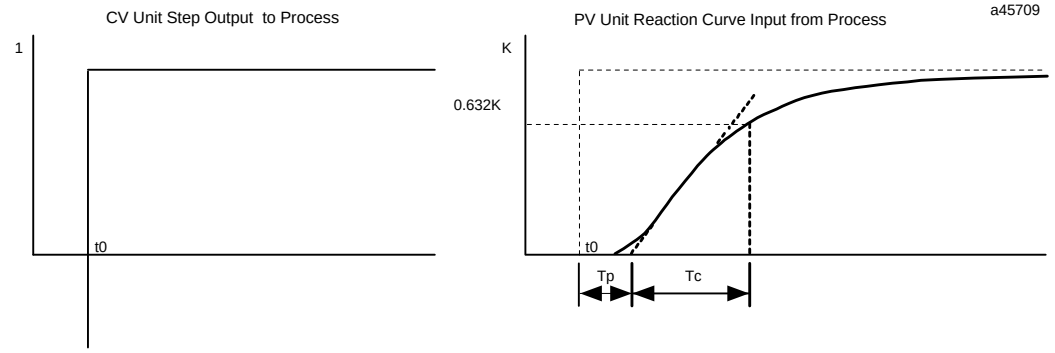
The PID loop gains, Kp, Ki and Kd, are determined by the characteristics of the process being controlled. Two key questions when setting up a PID loop are:

1. How big is the change in PV when we change CV by a fixed amount, or what is the open loop gain?
2. How fast does the system respond, or how quick does PV change after the CV output is stepped?

Many processes can be approximated by a process gain, first or second order lag and a pure time delay. In the frequency domain, the transfer function for a first order lag system with a pure time delay is:

PV(s)/CV(s) = G(s) = K * e **(–Tp s)/(1 + Tc s)

Plotting a step response at time t0 in the time domain provides an open loop unit reaction curve:



The following process model parameters can be determined from the PV unit reaction curve:

K	Process open loop gain = final change in PV/change in CV at time t0 (Note no subscript on K)
Tp	Process or pipeline time delay or dead time after t0 before the process output PV starts moving
Tc	First order Process time constant, time required after Tp for PV to reach 63.2% of the final PV

Usually the quickest way to measure these parameters is by putting the PID block in Manual mode and making a small step in CV output, by changing the Manual Command %Ref+13, and plotting the PV response over time. For slow processes, this can be done manually, but for faster processes a chart recorder or computer graphic data logging package will help. The CV step size should be large enough to cause an observable change in PV, but not so large that it disrupts the process being measured. A good size may be from 2 to 10% of the difference between the CV Upper and CV Lower Clamp values .

Setting User Parameters Including Tuning Loop Gains

As all PID parameters are totally dependent on the process being controlled, there are no predetermined values that will work, however, it is usually a simple, iterative procedure to find acceptable loop gain.

1. Set all the User Parameters to 0, then set the CV Upper and CV Lower Clamps to the highest and lowest CV expected. Set the Sample Period to the estimated process time constant (above)/10 to 100.
2. Put block in Manual mode and set Manual Command (%Ref+13) at different values to check if CV can be moved to Upper and Lower Clamp. Record PV value at some CV point and load it into SP.
3. Set a small gain, such as 100 * Maximum CV/Maximum PV, into Kp and turn off Manual mode. Step SP by 2 to 10% of the Maximum PV range and observe PV response. Increase Kp if PV step response is too slow or reduce Kp if PV overshoots and oscillates without reaching a steady value.
4. Once a Kp is found, start increasing Ki to get overshooting that dampens out to a steady value in 2 to 3 cycles. This may required reducing Kp. Also try different step sizes and CV operating points.

5. After suitable Kp and Ki gains are found, try adding Kd to get quicker responses to input changes providing it doesn't cause oscillations. Kd is often not needed and will not work with noisy PV.

6. Check gains over different SP operating points and add Dead Band and Minimum Slew Time if needed. Some Reverse Acting processes may need setting Config Word Error Sign or Polarity bits

Setting Loop Gains—Ziegler and Nichols Tuning Approach

Once the three process model parameters, K, Tp and Tc, are determined, they can be used to estimate initial PID loop gains. The following approach, developed by Ziegler and Nichols in the 1940's, is designed to provide good response to system disturbances with gains producing a amplitude ratio of 1/4. The amplitude ratio is the ratio of the second peak over the first peak in the closed loop response.

1. Calculate the Reaction rate:

   R = K/Tc

2. For Proportional control only, calculate Kp as

   Kp = 1/(R * Tp) = Tc/(K * Tp)

3. For Proportional and Integral control, use

   Kp = 0.9/(R * Tp) = 0.9 * Tc/(K * Tp)

   Ki = 0.3 * Kp/Tp

4. For Proportional, Integral and Derivative control, use

   Kp = G/(R * Tp) where G is from 1.2 to 2.0

   Ki = 0.5 * Kp/Tp

   Kd = 0.5 * Kp * Tp

5. Check that the Sample Period is in the range (Tp + Tc)/10 to (Tp + Tc)/1000

Another approach, the "Ideal Tuning" procedure, is designed to provide the best response to SP changes, delayed only by the Tp process delay or dead time.

Kp = 2 * Tc/(3 * K * Tp)

Ki = Tc

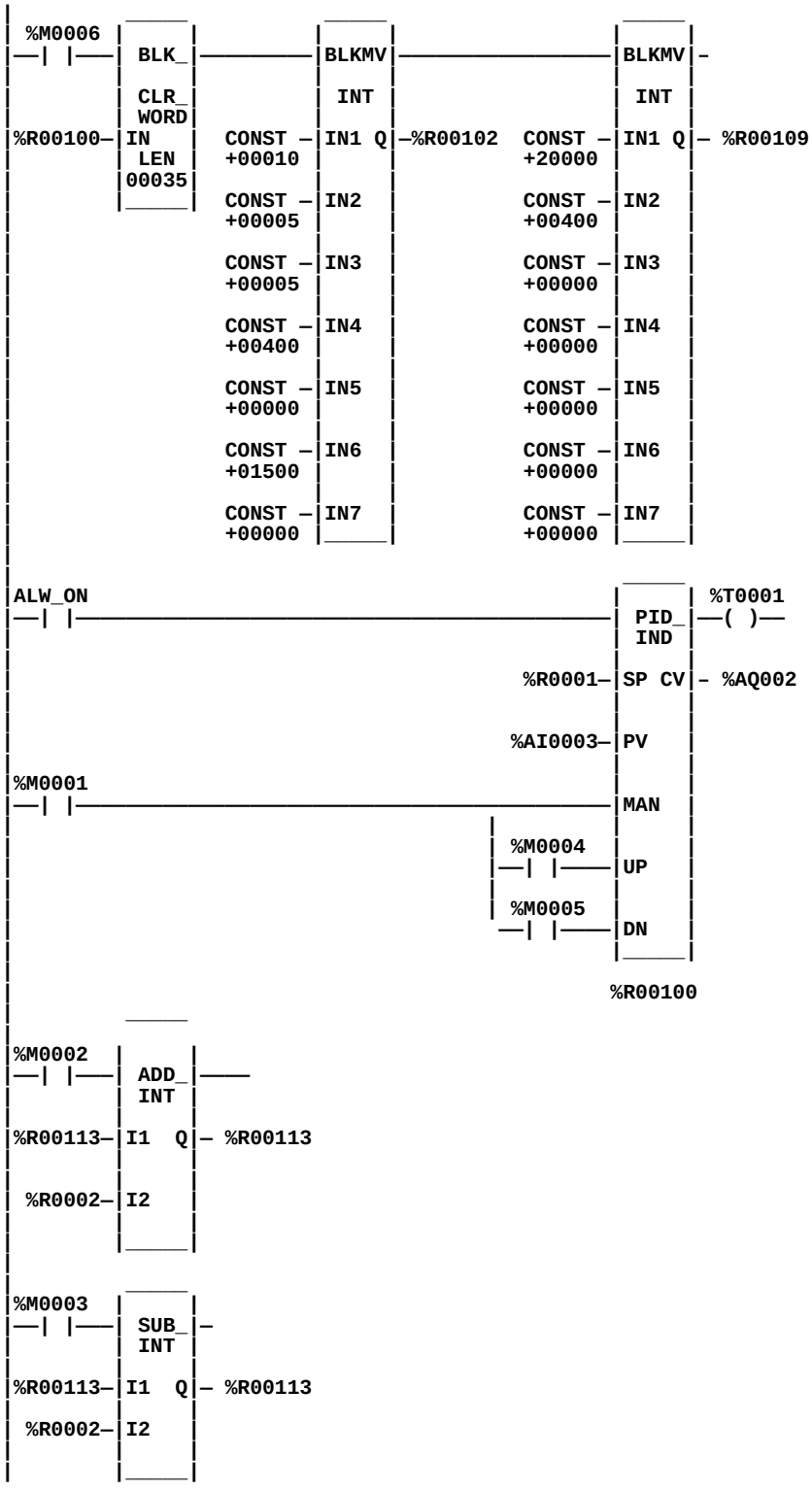
Kd = Ki/4 if Derivative term is used

Once initial gains are determined, they must be converted to integer User Parameters. To avoid scaling problems, the Process gain, K, should be calculated as a change in input PV Counts divided by the output step change in CV Counts and not in process PV or CV engineering units. All times should also be specified in seconds. Once Kp, Ki and Kd are determined, Kp and Kd can be multiplied by 100 and entered as integer while Ki can be multiplied by 1000 and entered into the User Parameter %RefArray.

Sample PID Call

The following example has a Sample Period of 100 millisecond, a Kp gain of 4.00 and a Ki gain of 1.500. The Set Point is stored in %R1 with the Control Variable output in %AQ2 and the Process Variable returned in %AI3. CV Upper and CV Lower Clamps must be set, in this case to 20000 and 400, and an optional small Dead Band of +5 and –5 has been included. The 40 word RefArray starts in %R100. Normally User Parameters are set in the RefArray with the PID Zoom key **F10**, but %M6 can be set to reinitialize the 14 words starting at %R102 (%Ref+2) from constants stored in logic.

The block can be switched to Manual mode with %M1 so that the Manual Command, %R113, can be adjusted. Bits %M4 or %M5 can be used to increase or decrease %R113 and the PID CV and integrator by 1 every 100 millisecond solution. For faster manual operation, bits %M2 and %M3 can be used to add or subtract the value in %R2 to/from %R113 every PLC sweep. The %T1 output is on when the PID is OK.


%M0006
BLK_
CLR_
WORD
%R00100
IN
LEN
00035
BLKMV
INT
CONST +00010
IN1
Q
%R00102
CONST +00005
IN2
CONST +00005
IN3
CONST +00400
IN4
CONST +00000
IN5
CONST +01500
IN6
CONST +00000
IN7
CONST +20000
CONST +00400
CONST +00000
CONST +00000
CONST +00000
%R00109
ALW_ON
PID_IND
%T0001
%R0001
SP CV
%AQ002
%AI0003
PV
%M0001
MAN
%M0004
UP
%M0005
DN
%R00100
%M0002
ADD_INT
%R00113
I1
I2
%R0002
%M0003
SUB_INT

Appendix A Instruction Timing

The Series 90-30, 90-20, and Micro PLCs support many different functions and function blocks. This appendix contains tables showing the memory size in bytes and the execution time in microseconds for each function. Memory size is the number of bytes required by the function in a ladder diagram application program.

Two execution times are shown for each function:

Execution Time	Description
Enabled	Time required to execute the function or function block when power flows into and out of the function. Typically, best-case times are when the data used by the block is contained in user RAM (word-oriented memory) and not in the ISCP cache memory (discrete memory).
Disabled	Time required to execute the function when power flows into the function or function block; however, it is in an inactive state, as when a timer is held in the reset state.

Note

Timers and counters are updated each time they are encountered in the logic, timers by the amount of time consumed by the last sweep and counters by one count.

Table A-1. Instruction Timing

Function Group	Function	Enabled				Disabled				Increment				Size
		311	313	331	340/41	311	313	331	340/41	311	313	331	340/41	
Timers	On-Delay Timer	146	81	80	42	105	39	38	21	–	–	–	–	15
	Off-Delay Timer	98	47	44	23	116	63	58	32	–	–	–	–	9
	Timer	122	76	75	40	103	54	53	30	–	–	–	–	15
Counters	Up Counter	137	70	69	36	130	63	62	33	–	–	–	–	11
	Down Counter	136	70	69	37	127	61	61	31	–	–	–	–	11
Math	Addition (INT)	76	47	46	24	41	0	1	0	–	–	–	–	13
	Addition (DINT)	90	60	60	34	41	1	0	0	–	–	–	–	13
	Subtraction (INT)	75	46	45	25	41	0	1	0	–	–	–	–	13
	Subtraction (DINT)	92	62	62	34	41	1	0	0	–	–	–	–	13
	Multiplication (INT)	79	49	50	28	41	0	1	0	–	–	–	–	13
	Multiplication (DINT)	108	80	101	43	41	1	0	0	–	–	–	–	13
	Division (INT)	79	51	50	27	41	0	1	0	–	–	–	–	13
	Division (DINT)	375	346	348	175	41	1	0	0	–	–	–	–	13
	Modulo Division (INT)	78	51	49	27	41	0	1	0	–	–	–	–	13
	Modulo Div (DINT)	134	103	107	54	41	1	0	0	–	–	–	–	13
	Square Root (INT)	153	124	123	65	42	0	1	0	–	–	–	–	9
	Square Root (DINT)	268	239	241	120	42	0	0	1	–	–	–	–	9
Relational	Equal (INT)	66	35	36	19	41	1	1	0	–	–	–	–	9
	Equal (DINT)	86	56	54	29	41	1	0	0	–	–	–	–	9
	Not Equal (INT)	67	39	35	22	41	1	1	0	–	–	–	–	9
	Not Equal (DINT)	81	51	51	28	41	1	0	0	–	–	–	–	9
	Greater Than (INT)	64	33	35	20	41	1	1	0	–	–	–	–	9
	Greater Than (DINT)	89	59	58	32	41	1	0	0	–	–	–	–	9
	Greater Than/Eq (INT)	64	36	34	19	41	1	1	0	–	–	–	–	9
	Greater Than/Eq (DINT)	87	58	57	30	41	1	0	0	–	–	–	–	9
	Less Than (INT)	66	35		19	41	1	1	0	–	–	–	–	9
	Less Than (DINT)	87	57		30	41	1	1	0	–	–	–	–	9
	Less Than/Equal (INT)	66	36	34	21	41	1	1	0	–	–	–	–	9
	Less Than/Equal (DINT)	86	57	56	31	41	1	1	0	–	–	–	–	9
	Range (INT)	92	58	54	29	46	1	0	1	–	–	–	–	15
	Range(DINT)	106	75	57	37	45	0	0	0	–	–	–	–	15
	Range(WORD)	93	60	54	29	0	0	0	0	–	–	–	–	15

Notes:
1. Time (in microseconds) is based on Release 5.01 of Logicmaster 90-30/20 software for Models 311, 313, 340, and 341 CPUs (Release 7 for the 331).
2. For table functions, increment is in units of length specified.; for bit operation functions, microseconds/bit.; for data move functions, microseconds/number of bits or words.
3. Enabled time for single length units of type %R, %AI, and %AQ.
4. COMMREQ time has been measured between CPU and HSC.
5. DOIO is the time to output values to discrete output module.
6. Where there is more than one possible case, the time indicated above represents the worst possible case.

Timing information for the Micro PLC: See the *Series 90™ Micro Programmable Logic Controller User's Manual* (GFK-1065B or later) for this information.

Timing information for 350 and 360 Series PLCs: See page A-6 and following.

Table A-1. Instruction Timing-Continued

Function Group	Function	Enabled				Disabled				Increment				Size
		311	313	331	340/41	311	313	331	340/41	311	313	331	340/41	
Bit Operation	Logical AND	67	37	37	22	42	0	0	1	–	–	–	–	13
	Logical OR	68	38	38	21	42	0	0	1	–	–	–	–	13
	Logical Exclusive OR	66	38	37	20	42	0	1	1	–	–	–	–	13
	Logical Invert, NOT	62	32	31	17	42	0	1	1	–	–	–	–	9
	Shift Bit Left	139	89	90	47	74	26	23	13	11.61	11.61	12.04	6.29	15
	Shift Bit Right	135	87	85	45	75	26	24	13	11.63	11.62	12.02	6.33	15
	Rotate Bit Left	156	127	126	65	42	1	1	0	11.70	11.78	12.17	6.33	15
	Rotate Bit Right	146	116	116	62	42	1	1	0	11.74	11.74	12.13	6.27	15
	Bit Position	102	72	49	38	42	1	0	0	–	–	–	–	13
	Bit Clear	68	38	35	21	42	1	1	1	–	–	–	–	13
	Bit Test	79	49	51	28	41	0	0	1	–	–	–	–	13
	Bit Set	67	37	37	20	42	0	0	0	–	–	–	–	13
	Masked Compare (WORD)	217	154	141	74	107	44	39	21	–	–	–	–	25
	Masked Compare (DWORD)	232	169	156	83	108	44	39	22	–	–	–	–	25
Data Move	Move (INT)	68	37	39	20	43	0	0	0	1.62	1.62	5.25	1.31	13
	Move (BIT)	94	62	64	35	42	0	0	0	12.61	12.64	12.59	6.33	13
	Move (WORD)	67	37	40	20	41	0	0	0	1.62	1.63	5.25	1.31	13
	Block Move (INT)	76	48	50	28	59	30	30	16	–	–	–	–	27
	Block Move (WORD)	76	48	49	29	59	29	28	15	–	–	–	–	27
	Block Clear	56	28	27	14	43	0	0	0	1.35	1.29	1.40	0.78	9
	Shift Register (BIT)	201	153	153	79	85	36	34	18	0.69	0.68	0.71	0.37	15
	Shift Register (WORD)	103	53	52	29	73	25	23	12	1.62	1.62	2.03	1.31	15
	Bit Sequencer	165	101	99	53	96	31	29	16	0.07	0.07	0.08	0.05	15
	COMM_REQ	1317	1272	1489	884	41	2	0	0	–	–	–	–	13
Table	Array Move													
	INT	230	201	177	104	72	41	40	20	1.29	1.15	10.56	2.06	21
	DINT	231	202	181	105	74	44	42	23	3.24	3.24	10.53	2.61	21
	BIT	290	261	229	135	74	43	42	23	–.03	–.03	-0.01	0.79	21
	BYTE	228	198	176	104	74	42	42	23	0.81	0.82	8.51	1.25	21
	WORD	230	201	177	104	72	41	40	20	1.29	1.15	10.56	2.06	21
	Search Equal													
	INT	197	158	123	82	78	39	37	20	1.93	1.97	2.55	1.55	19
	DINT	206	166	135	87	79	38	36	21	4.33	4.34	4.55	2.44	19
	BYTE	179	141	117	74	78	38	36	21	1.53	1.49	1.83	1.03	19
	WORD	197	158	123	82	78	39	37	20	1.93	1.97	2.55	1.55	19

Notes:
1. Time (in microseconds) is based on Release 5.01 of Logicmaster 90-30/20 software for Models 311, 313, 340, and 341 CPUs (Release 7 for the 331).
2. For table functions, increment is in units of length specified.; for bit operation functions, microseconds/bit.; for data move functions, microseconds/number of bits or words.
3. Enabled time for single length units of type %R, %AI, and %AQ.
4. COMMREQ time has been measured between CPU and HSC.
5. DOIO is the time to output values to discrete output module.
6. Where there is more than one possible case, the time indicated above represents the worst possible case.
7. For instructions that have an increment value, multiply the increment by (Length –1) and add that value to the base time.

Timing information for the Micro PLC: See the *Series 90™ Micro Programmable Logic Controller User's Manual* (GFK-1065B or later) for this information.

Timing information for 350 and 360 Series PLCs: See page A-6 and following.

Table A-1. Instruction Timing-Continued

Function Group	Function	Enabled				Disabled				Increment				Size
		311	313	331	340/41	311	313	331	340/41	311	313	331	340/41	
	Search Not Equal													
	INT	198	159	124	83	79	39	36	21	1.93	1.93	2.48	1.52	19
	DINT	201	163	132	84	79	37	35	21	6.49	6.47	6.88	3.82	19
	BYTE	179	141	117	73	79	38	36	19	1.54	1.51	1.85	1.05	19
	WORD	198	159	124	83	79	39	36	21	1.93	1.93	2.48	1.52	19
	Search Greater Than													
	INT	198	160	125	82	79	37	38	19	3.83	3.83	4.41	2.59	19
	DINT	206	167	135	88	78	38	36	20	8.61	8.61	9.03	4.88	19
	BYTE	181	143	118	73	79	37	36	19	3.44	3.44	3.75	2.03	19
	WORD	198	160	125	82	79	37	38	19	3.83	3.83	4.41	2.59	19
	Search Greater Than/Eq													
	INT	197	160	124	83	77	38	36	20	3.86	3.83	4.45	2.52	19
	DINT	205	167	136	87	80	39	36	21	8.62	8.61	9.02	4.87	19
	BYTE	180	142	118	75	79	37	37	20	3.47	3.44	3.73	2.00	19
	WORD	197	160	124	83	77	38	36	20	3.86	3.83	4.45	2.52	19
	Search Less Than													
	INT	199	159	124	84	78	38	36	20	3.83	3.86	4.48	2.48	19
	DINT	206	168	135	87	79	38	38	19	8.62	8.60	-1.36	4.88	19
	BYTE	181	143	119	75	80	38	37	20	3.44	3.44	3.75	2.00	19
	WORD	199	159	124	84	78	38	36	20	3.83	3.86	4.45	2.48	19
	Search Less Than/Equal													
	INT	200	158	124	82	79	38	37	21	3.79	3.90	4.45	2.55	19
	DINT	207	167	137	88	78	39	37	19	8.60	8.61	9.01	4.86	19
	BYTE	180	143	119	74	78	40	37	19	3.46	3.44	3.73	2.02	19
	WORD	200	158	124	82	79	38	37	21	3.79	3.90	4.45	2.55	19
Conversion	Convert to INT	74	46	39	25	42	1	1	1	–	–	–	–	9
	Convert to BCD–4	77	50	34	25	42	1	1	1	–	–	–	–	9

Notes:
1. Time (in microseconds) is based on Release 5.01 of Logicmaster 90-30/20 software for Models 311, 313, 340, and 341 CPUs (Release 7 for the 331).
2. For table functions, increment is in units of length specified.; for bit operation functions, microseconds/bit.; for data move functions, microseconds/number of bits or words.
3. Enabled time for single length units of type %R, %AI, and %AQ.
4. COMMREQ time has been measured between CPU and HSC.
5. DOIO is the time to output values to discrete output module.
6. Where there is more than one possible case, the time indicated above represents the worst possible case.
7. For instructions that have an increment value, multiply the increment by (Length –1) and add that value to the base time.

Timing information for the Micro PLC: See the *Series 90™ Micro Programmable Logic Controller User's Manual* (GFK-1065B or later) for this information.

Timing information for 350 and 360 Series PLCs: See page A-6 and following.

Table A-1. Instruction Timing-Continued

Function Group	Function	Enabled				Disabled				Increment				Size
		311	313	331	340/41	311	313	331	340/41	311	313	331	340/41	
Control	Call a Subroutine	155	93	192	85	41	0	0	0	–	–	–	–	7
	Do I/O	309	278	323	177	38	1	0	0	–	–	–	–	12
	PID – ISA Algorithm	1870	1827	1812	929	91	56	82	30	–	–	–	–	15
	PID – IND Algorithm	2047	2007	2002	1017	91	56	82	30	–	–	–	–	15
	End Instruction	–	–	–	–	–	–	–	–	–	–	–	–	–
	Service Request													
	# 6	93	54	63	45	41	2	0	0	–	–	–	–	9
	# 7 (Read)	–	37	309	161	–	2	0	0	–	–	–	–	9
	# 7 (Set)	–	37	309	161	–	2	0	0	–	–	–	–	9
	#14	447	418	483	244	41	2	0	0	–	–	–	–	9
	#15	281	243	165	139	41	2	0	0	–	–	–	–	9
	#16	131	104	115	69	41	2	0	0	–	–	–	–	9
	#18	–	56	300	180	–	2	0	0	–	–	–	–	9
	#23	1689	1663	1591	939	43	1	0	0	–	–	–	–	9
	#26//30*	1268	1354	6680	3538	42	0	0	0	–	–	–	–	9
	#29	–	–	55	41	–	–	1	0	–	–	–	–	9
	Nested MCR/ENDMCR Combined	135	73	68	39	75	25	21	12	–	–	–	–	8

*Service request #26/30 was measured using a high speed counter, 16-point output, in a 5-slot rack.

Notes:
1. Time (in microseconds) is based on Release 5.01 of Logicmaster 90-30/20 software for Models 311, 313, 340, and 341 CPUs (Release 7 for the 331).
2. For table functions, increment is in units of length specified.; for bit operation functions, microseconds/bit.; for data move functions, microseconds/number of bits or words.
3. Enabled time for single length units of type %R, %AI, and %AQ.
4. COMMREQ time has been measured between CPU and HSC.
5. DOIO is the time to output values to discrete output module.
6. Where there is more than one possible case, the time indicated above represents the worst possible case.
7. For instructions that have an increment value, multiply the increment by (Length –1) and add that value to the base time.

Timing information for the Micro PLC: See the *Series 90™ Micro Programmable Logic Controller User's Manual* (GFK-1065B or later) for this information.

Timing information for 350 and 360 Series PLCs: See page A-6 and following.

Table A-1. Instruction Timing-Continued

Function Group	Function	Enabled 350/351/36x	Disabled 350/351/36x	Increment 350/351/36x	Enabled 352	Disabled 352	Increment 352	Size
Timers	On-Delay Timer	4	6	–	4	5	–	15
	Timer	3	3	–	2	2	–	15
	Off-Delay Timer	3	3	–	3	2	–	15
Counters	Up Counter	1	3	–	2	2	–	13
	Down Counter	3	3	–	1	2	–	13
Math	Addition (INT)	2	0	–	1	0	–	13
	Addition (DINT)	2	0	–	2	0	–	19
	Addition (REAL)	52	0	–	33	0	–	17
	Subtraction (INT)	2	0	–	1	0	–	13
	Subtraction (DINT)	2	0	–	2	0	–	19
	Subtraction (REAL)	53	0	–	34	0	–	17
	Multiplication (INT)	21	0	–	21	0	–	13
	Multiplication (DINT)	24	0	–	24	0	–	19
	Multiplication (REAL)	68	1	–	38	1	–	17
	Division (INT)	22	0	–	22	0	–	13
	Division (DINT),	25	0	–	25	0	–	19
	Division (REAL)	82	2	–	36	2	–	17
	Modulo Division (INT)	21	0	–	21	0	–	13
	Modulo Div (DINT)	25	0	–	25	0	–	19
	Square Root (INT)	42	1	–	41	1	–	10
	Square Root (DINT)	70	0	–	70	0	–	13
	Square Root (REAL)	137	0	–	35	0	–	11
Trigonometric	SIN (REAL)	360	0	–	32	0	–	11
	COS (REAL)	319	0	–	29	0	–	11
	TAN (REAL)	510	1	–	32	1	–	11
	ASIN (REAL)	440	0	–	45	0	–	11
	ACOS (REAL)	683	0	–	63	0	–	11
	ATAN (REAL)	264	1	–	33	1	–	11
Logarithmic	LOG (REAL)	469	0	–	32	0	–	11
	LN (REAL)	437	0	–	32	0	–	11
Exponential	EXP	639	0	–	42	0	–	11
	EXPT	89	1	–	54	1	–	17
Radian Conversion	Convert RAD to DEG	65	1	–	32	1	–	11
	Convert DEG to RAD	59	0		32	0		11

Notes:
1. Time (in microseconds) is based on Release 7 of Logicmaster 90-30/20/Micro software for Model 351 and 352 CPUs.
2. For table functions, increment is in units of length specified.; for bit operation functions, microseconds/bit.; for data move functions, microseconds/number of bits or words.
3. Enabled time for single length units of type %R, %AI, and %AQ.
4. COMMREQ time has been measured between CPU and HSC.
5. DOIO is the time to output values to discrete output module.
6. Where there is more than one possible case, the time indicated above represents the worst possible case.

Table A-1. Instruction Timing-Continued

Function Group	Function	Enabled 350/351/36x	Disabled 350/351/36x	Increment 350/351/36x	Enabled 352	Disabled 352	Increment 352	Size
Relational	Equal (INT)	1	0	–	1	0	–	10
	Equal (DINT)	2	0	–	2	0	–	16
	Equal (REAL)	57	0	–	28	0	–	14
	Not Equal (INT)	1	0	–	1	0	–	10
	Not Equal (DINT)	1	0	–	1	0	–	16
	Not Equal (REAL)	62	0	–	31	0	–	14
	Greater Than (INT)	1	0	–	1	0	–	10
	Greater Than (DINT)	1	0	–	1	0	–	16
	Greater Than (REAL)	57	0	–	32	0	–	14
	Greater Than/Equal (INT)	1	0	–	1	0	–	10
	Greater Than/Equal (DINT)	1	0	–	1	0	–	10
	Greater Than/Equal (REAL)	57	1	–	31	1	–	14
	Less Than (INT)	1	0	–	1	0	–	10
	Less Than (DINT)	1	0	–	1	0	–	16
	Less Than (REAL)	58	1	–	36	1	–	14
	Less Than/Equal (INT)	1	0	–	1	0	–	10
	Less Than/Equal (DINT)	3	0	–	3	0	–	16
	Less Than/Equal (REAL)	37	0	–	37	0	–	14
	Range (INT)	2	1	–	2	1	–	13
	Range (DINT)	2	1	–	2	1	–	22
	Range (WORD)	1	0	–	1	0	–	13
Bit Operation	Logical AND	2	0	–	2	0	–	13
	Logical OR	2	0	–	2	0	–	13
	Logical Exclusive OR	1	0	–	1	0	–	13
	Logical Invert, NOT	1	0	–	1	0	–	10
	Shift Bit Left	31	1	1.37	31	1	1.37	16
	Shift Bit Right	28	0	3.03	28	0	3.03	16
	Rotate Bit Left	25	0	3.12	25	0	3.12	16
	Rotate Bit Right	25	0	4.14	25	0	4.14	16
	Bit Position	20	1	–	20	1	–	13
	Bit Clear	20	0	–	20	0	–	13
	Bit Test	20	0	–	20	0	–	13
	Bit Set	19	1	–	19	1	–	13
	Mask Compare (WORD)	52	0	–	52	0	–	25
	Mask Compare (DWORD)	50	0	–	49	0	–	25

Notes:
1. Time (in microseconds) is based on Release 7 of Logicmaster 90-30/20/Micro software for Model 351 and 352 CPUs.
2. For table functions, increment is in units of length specified.; for bit operation functions, microseconds/bit.; for data move functions, microseconds/number of bits or words.
3. Enabled time for single length units of type %R, %AI, and %AQ.
4. COMMREQ time has been measured between CPU and HSC.
5. DOIO is the time to output values to discrete output module.
6. Where there is more than one possible case, the time indicated above represents the worst possible case.
7. For instructions that have an increment value, multiply the increment by (Length –1) and add that value to the base time.

Table A-1. InstructionTiming-Continued

Function Group	Function	Enabled 350/351/36X	Disabled 350/351/36X	Increment 350/351/36X	Enabled 352	Disabled 352	Increment 352	Size
Data Move	Move (INT)	2	0	0.41	2	0	0.41	10
	Move (BIT)	28	0	4.98	28	0	4.98	13
	Move (WORD)	2	0	0.41	2	0	0.41	10
	Move (REAL)	24	1	0.82	24	1	0.82	13
	Block Move (INT)	2	0	–	2	0	–	28
	Block Move (WORD)	4	4	–	3	0	–	28
	Block Move (REAL)	41	0	–	41	0	–	13
	Block Clear	1	0	0.24	1	0	0.24	11
	Shift Register (BIT)	49	0	0.23	46	0	0.23	16
	Shift Register (WORD)	27	0	0.41	27	0	0.41	16
	Bit Sequencer	38	22	0.02	38	22	0.02	16
	COMM_REQ	765	0	–	765	0	–	13
Table	Array Move							
	INT	54	0	0.97	54	0	0.97	22
	DINT	54	0	0.81	54	0	0.81	22
	BIT	69	0	0.36	69	0	0.36	22
	BYTE	54	1	0.64	54	1	0.64	22
	WORD	54	0	0.97	54	0	0.97	22
	Search Equal							
	INT	37	0	0.62	37	0	0.62	19
	DINT	41	1	1.38	41	1	1.38	22
	BYTE	35	0	0.46	35	0	0.46	19
	WORD	37	0	0.62	37	0	0.62	19
	Search Not Equal							
	INT	37	0	0.62	37	0	0.62	19
	DINT	38	0	2.14	38	0	2.14	22
	BYTE	37	0	0.47	37	0	0.47	19
	WORD	37	0	0.62	37	0	0.62	19
	Search Greater Than							
	INT	37	0	1.52	37	0	1.52	19
	DINT	39	0	2.26	39	0	2.26	22
	BYTE	36	1	1.24	36	1	1.24	19
	WORD	37	0	1.52	37	0	1.52	19
	Search Greater Than/Equal							
	INT	37	0	1.48	37	0	1.48	19
	DINT	39	0	2.33	39	0	2.33	22
	BYTE	37	1	1.34	37	1	1.34	19
	WORD	37	0	1.48	37	0	1.48	19

Notes:
1. Time (in microseconds) is based on Release 7 of Logicmaster 90-30/20/Micro software for 350 and 360 Series CPUs.
2. For table functions, increment is in units of length specified.; for bit operation functions, microseconds/bit.; for data move functions, microseconds/number of bits or words.
3. Enabled time for single length units of type %R, %AI, and %AQ.
4. COMMREQ time has been measured between CPU and HSC.
5. DOIO is the time to output values to discrete output module.
6. Where there is more than one possible case, the time indicated above represents the worst possible case.
7. For instructions that have an increment value, multiply the increment by (Length –1) and add that value to the base time.

Table A-1. Instruction Timing-Continued

Function Group	Function	Enabled 350/351/36x	Disabled 350/351/36x	Increment 350/351/36x	Enabled 352	Disabled 352	Increment 352	Size
	Search Less Than							
	INT	37	0	1.52	37	0	1.52	19
	DINT	41	1	2.27	41	1	2.27	22
	BYTE	37	0	1.41	37	0	1.41	19
	WORD	37	0	1.52	37	0	1.52	19
	Search Less Than/Equal							
	INT	38	0	1.48	38	0	1.48	19
	DINT	40	1	2.30	40	1	2.30	22
	BYTE	37	0	1.24	37	0	1.24	19
	WORD	38	0	1.48	38	0	1.48	19
Conversion	Convert to INT	19	1	–	19	1	–	10
	Convert to BCD-4	21	1	–	21	1	–	10
	Convert to REAL	27	0	–	21	0	–	8
	Convert to WORD	28	1	–	30	1	–	11
	Truncate to INT	32	0	–	32	0	–	11
	Truncate to DINT	63	0	–	31	0	–	11
Control	Call a Subroutine	72	1	–	73	1	–	7
	Do I/O	114	1	–	115	1	–	13
	PID – ISA Algorithm*	162	34	–	162	34	–	16
	PID – IND Algorithm*	146	34	–	146	34	–	16
	End Instruction	–	–	–	–	–	–	–
	Service Request							
	#6	22	1	–	22	1	–	10
	#7 (Read)	75	1	–	75	1	–	10
	#7 (Set)	75	1	–	75	1	–	10
	#14	121	1	–	121	1	–	10
	#15	46	1	–	46	1	–	10
	#16	36	1	–	36	1	–	10
	#18	261	1	–	261	1	–	10
	#23	426	0	–	426	0	–	10
	#26//30**	2260	1	–	2260	1	–	10
	#29	20	0	–	20	0	–	10
	#43							
	Nested MCR/ENDMCR Combined	1	1	–	1	1	–	4
	Sequential Event Recorder							

***The PID times shown above are based on the 6.5 release of the 351 CPU.

****Service request #26/30 was measured using a high speed counter, 16-point output, in a 5-slot rack.

Notes:
1. Time (in microseconds) is based on Release 7 of Logicmaster 90-30/20/Micro software for 350 and 360 Series CPUs.
2. For table functions, increment is in units of length specified.; for bit operation functions, microseconds/bit.; for data move functions, microseconds/number of bits or words.
3. Enabled time for single length units of type %R, %AI, and %AQ.
4. COMMREQ time has been measured between CPU and HSC.
5. DOIO is the time to output values to discrete output module.
6. Where there is more than one possible case, the time indicated above represents the worst possible case.
7. For instructions that have an increment value, multiply the increment by (Length –1) and add that value to the base time.

Instruction Sizes for 350 and 360 Series CPUs

Memory size is the number of bytes required by the instruction in a ladder diagram application program. 350 and 360 Series CPUs require three (3) bytes for most standard boolean functions—see Table A-2.

Table A-2. Instruction Sizes for 350 and 360 Series CPUs

Function	Size
No operation	1
Pop stack and AND to top	1
Pop stack and OR to top	1
Duplicate top of stack	1
Pop stack	1
Initial stack	1
Label	5
Jump	5
All other instructions	3
Function blocks—see Table A-1	–

Boolean Execution Speed

The execution times of coils and contacts are shown below. These times represent the time used for each coil or contact in your RLD program.

Table A-3. Boolean Execution Speeds

Model 350 and 360 Series	0.22 milliseconds per 1,000 boolean contacts/coils
Model 340/341	0.3 milliseconds per 1,000 boolean contacts/coils
Model 331	0.4 milliseconds per 1,000 boolean contacts/coils
Model 313/323	0.6 milliseconds per 1,000 boolean contacts/coils
Model 311	18.0 milliseconds per 1,000 boolean contacts/coils

Appendix B Interpreting Fault Tables

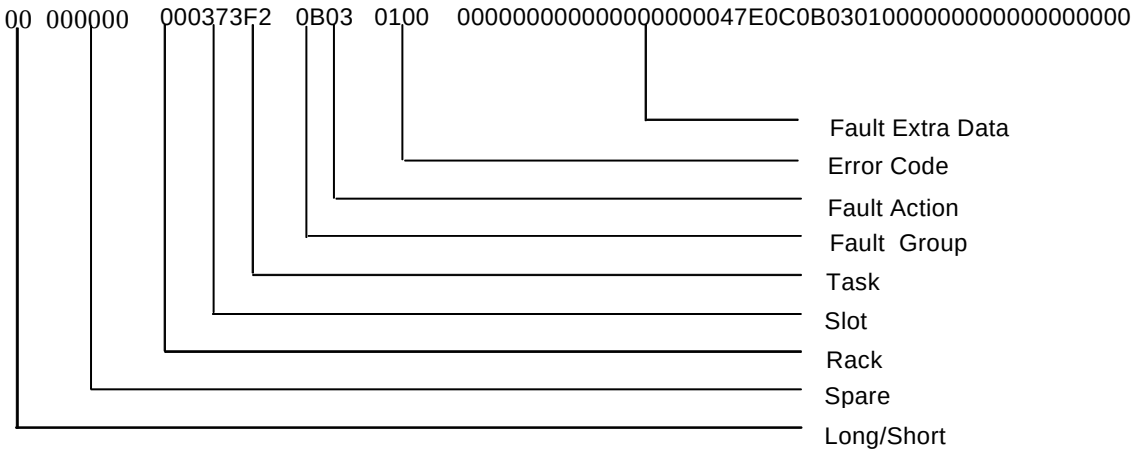
The Series 90-30 PLCs maintain two fault tables, the I/O fault table for faults generated by I/O devices (including I/O controllers) and the PLC fault table for internal PLC faults. The information in this appendix will enable you to interpret the message structure format when reading these fault tables. Both tables contain similar information.

- The PLC fault table contains:
  - Fault location.
  - Fault description.
  - Date and time of fault.
- The I/O fault table contains:
  - Fault location.
  - Reference address.
  - Fault category.
  - Fault type.
  - Date and time of fault.

PLC Fault Table

Access the PLC fault table through your programming software.

The following diagram identifies each field in the fault entry for the System Configuration Mismatch fault displayed above:



The System Configuration Mismatch fault entry is explained below. (All data is in hexadecimal.)

Field	Value	Description
Long/Short	00	This fault contains 8 bytes of fault extra data.
Rack	00	Main rack (rack 0).
Slot	03	Slot 3.
Task	44	
Fault Group	0B	System Configuration Mismatch fault.
Fault Action	03	FATAL fault.
Error Code	01	

The following paragraphs describe each field in the fault entry. Included are tables describing the range of values each field may have.

Long/Short Indicator

This byte indicates whether the fault contains 8 bytes or 24 bytes of fault extra data.

Type	Code	Fault Extra Data
Short	00	8 bytes
Long	01	24 bytes

Spare

These six bytes are pad bytes, used to make the PLC fault table entry exactly the same length as the I/O fault table entry.

Rack

The rack number ranges from 0 to 7. Zero is the main rack, containing the PLC. Racks 1 through 7 are expansion racks, connected to the PLC through an expansion cable.

Slot

The slot number ranges from 0 to 9. The PLC CPU always occupies slot 1 in the main rack (rack 0).

Task

The task number ranges from 0 to +65,535. Sometimes the task number gives additional information for PLC engineers; typically, the task can be ignored.

PLC Fault Group

Fault group is the highest classification of a fault. It identifies the general category of the fault.

Table B-1 lists the possible fault groups in the PLC fault table.

The last non-maskable fault group, ***Additional PLC Fault Codes***, is declared for the handling of new fault conditions in the system without the PLC having to specifically know the alarm codes. All unrecognized PLC-type alarm codes belong to this group.

Table B-1. PLC Fault Groups

Group Number		Group Name	Fault Action
Decimal	Hexadecimal		
1	1	Loss of, or missing, rack.	Fatal
4	4	Loss of, or missing, option module.	Diagnostic
5	5	Addition of, or extra, rack.	Diagnostic
8	8	Addition of, or extra, option module.	Diagnostic
11	B	System configuration mismatch.	Fatal
12	C	System bus error.	Diagnostic
13	D	PLC CPU hardware failure.	Fatal
14	E	Non-fatal module hardware failure.	Diagnostic
16	10	Option module software failure.	Diagnostic
17	11	Program block checksum failure.	Fatal
18	12	Low battery signal.	Diagnostic
19	13	Constant sweep time exceeded.	Diagnostic
20	14	PLC system fault table full.	Diagnostic
21	15	I/O fault table full.	Diagnostic
22	16	User Application fault.	Diagnostic
–	–	Additional PLC fault codes.	As specified
128	80	System bus failure.	Fatal
129	81	No user’s program on power-up.	Informational
130	82	Corrupted user RAM detected.	Fatal
132	84	Password access failure.	Informational
135	87	PLC CPU software failure.	Fatal
137	89	PLC sequence-store failure.	Fatal

Fault Action

Each fault may have one of three actions associated with it. These fault actions are fixed on the Series 90-30 PLC and cannot be changed by the user.

Table B-2. PLC Fault Actions

Fault Action	Action Taken by CPU	Code
Informational	Log fault in fault table.	1
Diagnostic	Log fault in fault table. Set fault references.	2
Fatal	Log fault in fault table. Set fault references. Go to STOP mode.	3

Error Code

The error code further describes the fault. Each fault group has its own set of error codes. Table B-3 shows error codes for the PLC Software Error Group (Group 87H).

Table B-3. Alarm Error Codes for PLC CPU Software Faults

Decimal	Hexadecimal	Name
20	14	Corrupted PLC Program Memory.
39	27	Corrupted PLC Program Memory.
82	52	Backplane Communications Failed.
90	5A	User Shut Down Requested.
All others		PLC CPU Internal System Error.

Table B-4 shows the error codes for all the other fault groups.

Table B-4. Alarm Error Codes for PLC Faults

Decimal	Hexadecimal	Name
PLC Error Codes for Loss of Option Module Group		
44	2C	Option Module Soft Reset Failed.
45	2D	Option Module Soft Reset Failed.
255	FF	Option Module Communication Failed.
Error Codes for Reset of, Addition of, or Extra Option Module Group		
2	2	Module Restart Complete.
	All others	Reset of, Addition of, or Extra Option Module.
Error Codes for Option Module Software Failure Group		
1	1	Unsupported Board Type.
2	2	COMREQ – mailbox full on outgoing message that starts the COMREQ.
3	3	COMREQ – mailbox full on response.
5	5	Backplane Communications with PLC; Lost Request.
11	B	Resource (alloc, tbl ovrflw, etc.) error.
13	D	User program error.
401	191	Module Software Corrupted; Requesting Reload.
Error Codes for System Configuration Mismatch Group		
8	8	Analog Expansion Mismatch.
10	A	Unsupported Feature.
23	17	Program exceeds memory limits.
Error Codes for System Bus Error Group		
All others		System Bus Error.
Error Codes for Program Block Checksum Group		
3	3	Program or program block checksum failure.
Error Codes for Low Battery Signal		
0	0	Failed battery on PLC CPU or other module.
1	1	Low battery on PLC CPU or other module.
Error Codes for User Application Fault Group		
2	2	PLC Watchdog Timer Timed Out .
5	5	COMREQ – WAIT mode not available for this command.
6	6	COMREQ – Bad Task ID.
7	7	Application Stack Overflow.
Error Codes for System Bus Failure Group		
1	1	Operating system.
Error Codes for Corrupted User RAM on Powerup Group		
1	1	Corrupted User RAM on Power-up.
2	2	Illegal Boolean Opcode Detected.
3	3	PLC_ISCP_PC_OVERFLOW.
4	4	PRG_SYNTAX_ERR.
Error Codes for PLC CPU Hardware Faults		
All codes		PLC CPU Hardware Failure.

Fault Extra Data

This field contains details of the fault entry. An example of what data may be present are:

Corrupted User RAM Group:

Four of the error codes in the System Configuration Mismatch group supply fault extra data:

Table B-5. PLC Fault Data - Illegal Boolean Opcode Detected

Fault Extra Data	Model Number Mismatch
[0]	ISCP Fault Register Contents
[1]	Bad OPCODE
[2,3]	ISCP Program Counter
[4,5]	Function Number

For a RAM failure in the PLC CPU (one of the faults reported as a PLC CPU hardware failure), the address of the failure is stored in the first four bytes of the field.

PLC CPU Hardware Failure (RAM Failure):

PLC Fault Time Stamp

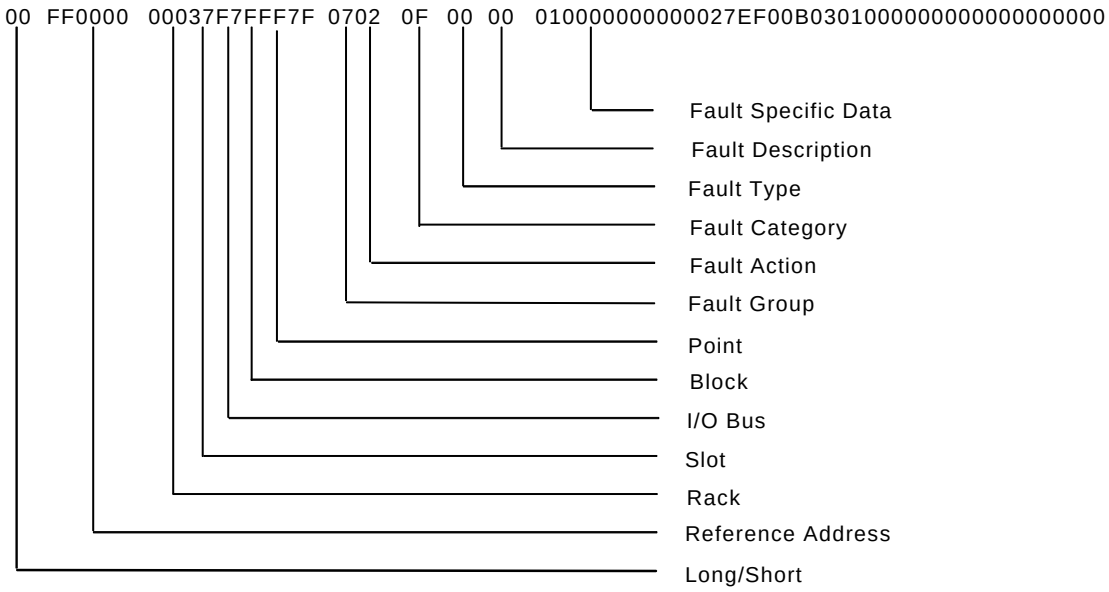
The six-byte time stamp is the value of the system clock when the fault was recorded by the PLC CPU. (Values are coded in BCD format.)

Table B-6. PLC Fault Time Stamp

Byte Number	Description
1	Seconds
2	Minutes
3	Hours
4	Day of the month
5	Month
6	Year

I/O Fault Table

The following diagram identifies the hexadecimal information displayed in each field in the fault entry.



The following paragraphs describe each field in the I/O fault table. Included are tables describing the range of values each field may have.

Long/Short Indicator

This byte indicates whether the fault contains 5 bytes or 21 bytes of fault specific data.

Table B-7. I/O Fault Table Format Indicator Byte

Type	Code	Fault Specific Data
Short	02	5 bytes
Long	03	21 bytes

Reference Address

Reference address is a three-byte address containing the I/O memory type and location (or offset) in that memory which corresponds to the point experiencing the fault. Or, when a Genius block fault or integral analog module fault occurs, the reference address refers to the first point on the block where the fault occurred.

Table B-8. I/O Reference Address

Byte	Description	Range
0	Memory Type	0 – FF
1–2	Offset	0 – 7FF

The memory type byte is one of the following values.

Table B-9. I/O Reference Address Memory Type

Name	Value (Hexadecimal)
Analog input	0A
Analog output	0C
Analog grouped	0D
Discrete input	10 or 46
Discrete output	12 or 48
Discrete grouped	1F

I/O Fault Address

The I/O fault address is a six-byte address containing rack, slot, bus, block, and point address of the I/O point which generated the fault. The point address is a word; all other addresses are one byte each. All five values may not be present in a fault.

When an I/O fault address does not contain all five addresses, a 7F hex appears in the address to indicate where the significance stops. For example, if 7F appears in the bus byte, then the fault is a module fault. Only rack and slot values are significant.

Rack

The rack number ranges from 0 to 7. Zero is the main rack, i.e., the one containing the PLC. Racks 1 through 7 are expansion racks.

Slot

The slot number ranges from 0 to 9. The PLC CPU always occupies slot 1 in the main rack (rack 0).

Point

Point ranges from 1 to 1024 (decimal). It tells which point on the block has the fault when the fault is a point-type fault.

I/O Fault Group

Fault group is the highest classification of a fault. It identifies the general category of the fault. The fault description text displayed by Logicmaster 90-30/20/Micro software is based on the fault group and the error codes.

Table B-10 lists the possible fault groups in the I/O fault table. Group numbers less than 80 (Hex) are maskable faults.

The last non-maskable fault group, ***Additional I/O Fault Codes***, is declared for the handling of new fault conditions in the system without the PLC having to specifically know the alarm codes. All unrecognized I/O-type alarm codes belong to this group.

Table B-10. I/O Fault Groups

Group Number	Group Name	Fault Action
3	Loss of, or missing, I/O module.	Diagnostic
7	Addition of, or extra, I/O module.	Diagnostic
9	IOC or I/O bus fault.	Diagnostic
A	I/O module fault.	Diagnostic
–	Additional I/O fault codes.	As specified

I/O Fault Action

The fault action specifies what action the PLC CPU should take when a fault occurs. Table B-11 lists possible fault actions.

Table B-11. I/O Fault Actions

Fault Action	Action Taken by CPU	Code
Informational	Log fault in fault table.	1
Diagnostic	Log fault in fault table. Set fault references.	2
Fatal	Log fault in fault table. Set fault references. Go to STOP mode.	3

I/O Fault Specific Data

An I/O fault table entry may contain up to 5 bytes of I/O fault specific data.

Symbolic Fault Specific Data

Table B-12 lists data that is required for block circuit configuration.

Table B-12. I/O Fault Specific Data

Decimal Number	Hex Code	Description
Circuit Configuration		
	1 2 3	Circuit is an input – tristate. Circuit is an input. Circuit is an output.

Fault Actions for Specific Faults

Forced/unforced circuit faults are reported as informational faults. All others are diagnostic or fatal.

The model number mismatch, I/O type mismatch and non-existent I/O module faults are reported in the PLC fault table under the System Configuration Mismatch group. They are not reported in the I/O fault table.

I/O Fault Time Stamp

The six-byte time stamp is the value of the system clock when the fault was recorded by the PLC CPU. Values are coded in BCD format.

Table B-13. I/O Fault Time Stamp

Byte Number	Description
1	Seconds
2	Minutes
3	Hours
4	Day of the month
5	Month
6	Year

Appendix C Instruction Mnemonics

In Program Display/Edit mode, you can quickly enter or search for a programming instruction by typing the ampersand (&) character followed by the instruction's mnemonic. For some instructions, you can also specify a reference address or nickname, a label, or a location reference address.

This appendix lists the mnemonics of the programming instructions for Logicmaster 90-30/20/Micro programming software. The complete mnemonic is shown in column 3 of this table, and the shortest entry you can make for each instruction is listed in column 4.

At any time during programming, you can display a help screen with these mnemonics by pressing the ALT and I keys.

Function Group	Instruction	Mnemonic						
		All	INT	DINT	BIT	BYTE	WORD	REAL
Contacts	Any Contact	&CON	&CON					
	Normally Open Contact	&NOCON	&NOCON					
	Normally Closed Contact	&NCCON	&NCCON					
	Continuation Contact	&CONC	&CONC					
Coils	Any Coil	&COI	&COI					
	Normally Open Coil	&NOCOI	&NOCOI					
	Negated Coil	&NCCOI	&NCCOI					
	Positive Transition Coil	&PCOI	&PCOI					
	Negative Transition Coil	&NCOI	&NCOI					
	SET Coil	&SL	&SL					
	RESET Coil	&RL	&RL					
	Retentive SET Coil	&SM	&SM					
	Retentive RESET Coil	&RM	&RM					
	Retentive Coil	&NOM	&NOM					
	Negated Retentive Coil	&NCM	&NCM					
	Continuation Coil	&COILC	&COILC					
Links	Horizontal Link	&HO	&HO					
	Vertical Link	&VE	&VE					
Timers	On Delay Timer	&ON	&ON					
	Elapsed Timer	&TM	&TM					
	Off Delay Timer	&OF	&OF					
Counters	Up Counter	&UP	&UP					
	Down Counter	&DN	&DN					

Function Group	Instruction	Mnemonic							
		All	**BCD-4**	**INT**	**DINT**	**BIT**	**BYTE**	**WORD**	**REAL**
Math	Addition	&AD		&AD_I	&AD_DI				&AD_R
	Subtraction	&SUB		&SUB_I	&SUB_DI				&SUB_R
	Multiplication	&MUL		&MUL_I	&MUL_DI				&MUL_R
	Division	&DIV		&DIV_I	&DIV_DI				&DIV_R
	Modulo	&MOD		&MOD_I	&MOD_DI				&MOD_R&SQ_R
	Square Root	&SQ		&SQ_I	&SQ_DI				
	Sine	&SIN							
	Cosine	&COS							
	Tangent	&TAN							
	Inverse Sine	&ASIN							
	Inverse Cosine	&ACOS							
	Inverse Tangent	&ATAN							
	Base 10 Logarithm	&LOG							
	Natural Logarithm	&LN							
	Power of e	&EXP							
	Power of x	&EXPT							
Relational	Equal	&EQ		&EQ_I	&EQ_DI				&EQ_R
	Not Equal	&NE		&NE_I	&NE_DI				&NE_R
	Greater Than	&GT		&GT_I	&GT_DI				&GT_R
	Greater or Equal	&GE		&GE_I	&GE_DI				&GE_R
	Less Than	&LT		&LT_I	&LT_DI				&LT_R
	Less Than or Equal	&LE		&LE_I	&LE_DI				&LE_R
Bit Operation	AND	&AN						&AN_W	
	OR	&OR						&OR_W	
	Exclusive OR	&XO						&XO_W	
	NOT	&NOT						&NOT_W	
	Bit Shift Left	&SHL						&SHL_W	
	Bit Shift Right	&SHR						&SHR_W	
	Bit Rotate Left	&ROL						&ROL_W	
	Bit Rotate Right	&ROR						&ROR_W	
	Bit Test	&BT						&BT_W	
	Bit Set	&BS						&BS_W	
	Bit Clear	&BCL						&BCL_W	
	Bit Position	&BP						&BP_W	
	Masked Compare	&MCM						&MCM_W	
Conversion	Convert to Integer	&TO_INT	&TO_INT_BCD4	&MOV					
	Convert to Double Integer	&TO_DINT		&BLKM					&BCD4_R
	Convert to BCD–4	&BCD4		&BLKC	&TO_REAL_DI				
	Convert to REAL	&TO_REAL		&SHF				&TO_REAL_W	
	Convert to WORD	&TO_W		&BI					
	Truncate to Integer	&TRINT		&COMMR					
	Truncate to Double Integer	&TRDINT							

Function Group	Instruction	Mnemonic						
		All	INT	DINT	BIT	BYTE	WORD	REAL
Data Move	Move	&MOV	&MOV_I		&MOV_BI		&MOV_W	&MOV_R
	Block Move	&BLKM	&BLKM_I				&BLKM_W	&BLKM_R
	Block Clear	&BLKC						
	Shift Register	&SHF			&SHF_BI			
	Bit Sequencer	&BI					&AR_W	
	Communications Request	&COMMR						
Table	Array Move	&AR	&AR_I	&AR_DI	&AR_BI	&AR_BY	&AR_W	
	Search Equal	&SRCHE	&SRCHE_I	&SRCHE_DI		&SRCHE_BY	&SRCHE_W	
	Search Not Equal	&SRCHN	&SRCHN_I	&SRCHN_DI		&SRCHN_BY	&SRCHN_W	
	Search Greater Than	&SRCHGT	&SRCHGT_I	&SRCHGT_DI		&SRCHGT_BY	&SRCHGT_W	
	Search Greater Than or Equal	&SRCHGE	&SRCHGE_I	&SRCHGE_DI		&SRCHGE_BY	&SRCHGE_W	
	Search Less Than	&SRCHLT	&SRCHLT_I	&SRCHLT_DI		&SRCHLT_BY	&SRCHLT_W	
	Search Less Than or Equal	&SRCHLE	&SRCHLE_I	&SRCHLE_DI		&SRCHLE_BY	&SRCHLE_W_	
Control	Call a Subroutine	&CA						
	Do I/O	&DO						
	SER	&SER						
	PID – ISA Algorithm	&PIDIS						
	PID – IND Algorithm	&PIDIN						
	SFC Reset	&SFCR						
	End	&END						
	Rung Explanation	&COMME						
	System Services Request	&SV						
	Master Control Relay	&MCR						
	End Master Control Relay	&ENDMCR						
	Nested Master Control Relay	&MCRN						
	Nested End Master Cntl Relay	&ENDMCRN						
	Jump	&JUMP						
	Nested Jump	&JUMPN						
	Label	&LABEL						
	Nested Label	&LABELN						

Appendix D Key Functions

This appendix lists the keyboard functions that are active in the software environment. This information may also be displayed on the programmer screen by pressing ALT-K to access key help.

Key Sequence	Description	Key Sequence	Description
Keys Available Throughout the Software Package			
ALT-A	Abort.	CTRL-Break	Exit package.
ALT-C	Clear field.	Esc	Zoom out.
ALT-M	Change Programmer mode.	CTRL-Home	Previous command-line contents.
ALT-R	Change PLC Run/Stop state.	CTRL-End	Next command-line contents.
ALT-E	Toggle status area.	CTRL- ←	Cursor left within the field.
ALT-J	Toggle command line.	CTRL-→	Cursor right within the field.
ALT-L	List directory files.	CTRL-D	Decrement reference address.
ALT-P	Print screen.	CTRL-U	Increment reference address.
ALT-H	Help.	Tab	Change/increment field contents.
ALT-K	Key help.	Shift-Tab	Change/decrement field contents.
ALT-I	Instruction mnemonic help.	Enter	Accept field contents.
ALT-N	Toggle display options.	CTRL-E	Display last system error.
ALT-T	Start Teach mode.	F12 or Keypad -	Toggle discrete reference.
ALT-Q	Stop Teach mode.	F11 or Keypad *	Override discrete reference.
ALT-n	Playback file n (n = 0 thru 9).		
Keys Available in the Program Editor Only			
ALT-B	Toggle text editor bell.	Keypad +	Accept rung.
ALT-D	Delete rung element/Delete rung.	Enter	Accept rung.
ALT-S	Store block to PLC and disk.	CTRL-PgUp	Previous rung.
ALT-X	Display zoom level.	CTRL-PgDn	Next rung.
ALT-U	Update disk.	~	Horizontal shunt.
ALT-V	Variable table window.	|	Vertical shunt.
ALT-F2	Go to operand reference table.	Tab	Go to the next operand field.
Special Keys			
ALT-O	Password override. Available only on the Password screen in the configuration software.		

The Help card on the next page contains a listing of the key help and also the instruction mnemonics help text for Logicmaster 90-30/20/Micro software. This card is printed in triplicate and is perforated for easier removal from the manual.

Print side 1 of GFJ-055C on this page.

Print side 2 of GFJ-055C on this page.

Appendix E Using Floating-Point Numbers

There are a few considerations you need to understand when using floating-point numbers. The first section discusses these general considerations. Refer to page E-5 and following for instructions on entering and displaying floating-point numbers.

Note

Floating-point capabilities are ***only*** supported on the 350 and 360 series CPUs, Release 9 or later, or on all releases of CPU352.

Floating-Point Numbers

The programming software provides the ability to edit, display, store, and retrieve numbers with real values. Some functions operate on floating-point numbers. However, to use floating-point numbers with the programming software, you must have a 350 or 360 series CPU (see Note above). Floating-point numbers are represented in decimal scientific notation, with a display of six significant digits.

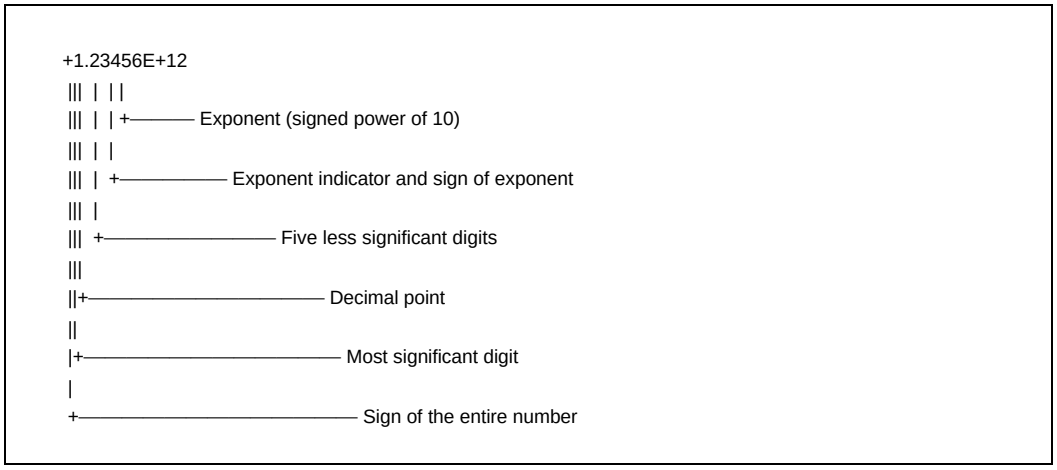
Note

In this manual, the terms “floating-point” and “real” are used interchangeably to describe the floating-point number display/entry feature of the programming software.

The following format is used. For numbers in the range 9999999 to .0001, the display has no exponent and up to six or seven significant digits. For example:

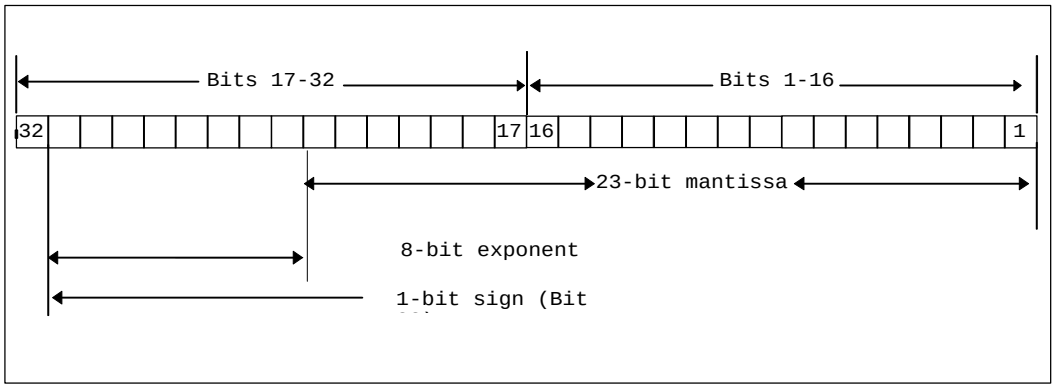
Entered	Displayed	Description
.000123456789	+.0001234567	Ten digits, six or seven significant.
–12.345e-2	–.1234500	Seven digits, six or seven significant.
1234	+1234.000	Seven digits, six or seven significant.

Outside the range listed above, only six significant digits are displayed and the display has the form:

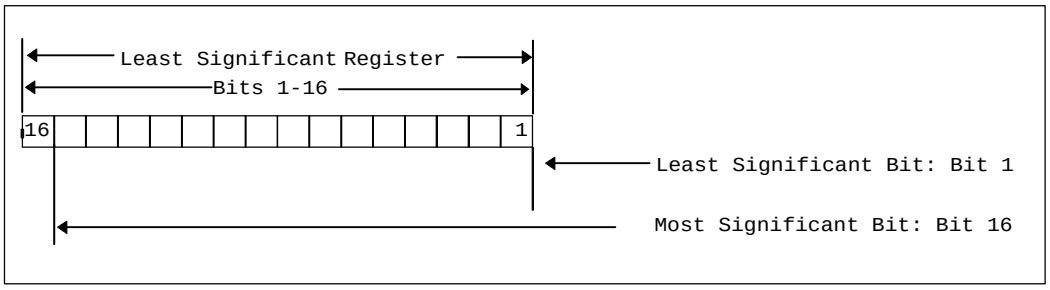


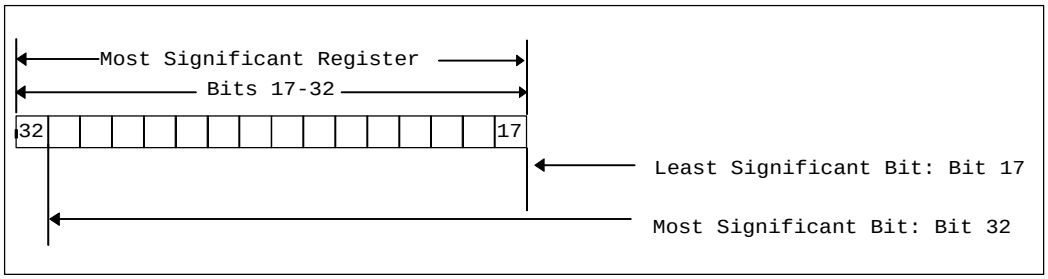
Internal Format of Floating-Point Numbers

Floating-point numbers are stored in single precision IEEE-standard format. This format requires 32 bits, which translates to two (adjacent) 16-bit PLC registers. The encoding of the bits is diagrammed below.



Register use by a single floating-point number is diagrammed below. In this diagram, if the floating-point number occupies registers R5 and R6, for example, then R5 is the least significant register and R6 is the most significant register.





Values of Floating-Point Numbers

Use the following table to calculate the value of a floating-point number from the binary number stored in two registers.

Exponent (e)	Mantissa (f)	Value of Floating Point Number
255	Non-zero	Not a valid number (NaN).
255	0	$-1^s * \infty$
0 < e < 255	Any value	$-1^s * 2^{e-127} * 1.f$
0	Non-zero	$-1^s * 2^{-126} * 0.f$
0	0	0

f = the mantissa. The mantissa is a binary fraction.
e = the exponent. The exponent is an integer E such that E+127 is the power of 2 by which the mantissa must be multiplied to yield the floating-point value.
s = the sign bit.
* = the multiplication operator.

For example, consider the floating-point number 12.5. The IEEE floating-point binary representation of the number is:

01000001 01001000 00000000 00000000

or 41480000 hex in hexadecimal form. The most significant bit (the sign bit) is zero (s=0). The next eight most significant bits are 10000010, or 130 decimal (e=130).

The mantissa is stored as a decimal binary number with the decimal point preceding the most significant of the 23 bits. Thus, the most significant bit in the mantissa is a multiple of 2^{-1}, the next most significant bit is a multiple of 2^{-2}, and so on to the least significant bit, which is a multiple of 2^{-23}. The final 23 bits (the mantissa) are:

1001000 00000000 00000000

The value of the mantissa, then, is .5625 (that is, $2^{-1} + 2^{-4}$).

Since e > 0 and e < 255, we use the third formula in the table above:

$$\begin{aligned} \text{number} &= -1^s * 2^{e-127} * 1.f \\ &= -1^0 * 2^{130-127} * 1.5625 \\ &= 1 * 2^3 * 1.5625 \\ &= 8 * 1.5625 \\ &= 12.5 \end{aligned}$$

Thus, you can see that the above binary representation is correct.

The range of numbers that can be stored in this format is from ± 1.401298E–45 to ± 3.402823E+38 and the number zero.

Entering and Displaying Floating-Point Numbers

In the mantissa, up to six or seven significant digits of precision may be entered and stored; however, the programming software will display only the first six of these digits. The mantissa may be preceded by a positive or negative sign. If no sign is entered, the floating-point number is assumed to be positive.

If an exponent is entered, it must be preceded by the letter **E** or **e**, and the mantissa must contain a decimal point to avoid mistaking it for a hexadecimal number. The exponent may be preceded by a sign; but, if none is provided, it is assumed to be positive. If no exponent is entered, it is assumed to be zero. No spaces are allowed in a floating-point number.

To provide ease-of-use, several formats are accepted in both command-line and field data entry. These formats include an integer, a decimal number, or a decimal number followed by an exponent. These numbers are converted to a standard form for display once the user has entered the data and pressed the **Enter** key.

Examples of valid floating-point number entries and their normalized display are shown below.

Entered	Displayed
250	+250,0000
+4	+4.000000
–2383019	–2383019.
34.	+34.00000
–.0036209	–.003620900
12.E+9	+1.20000E+10
–.0004E–11	–4.00000E–15
731.0388	+731.0388
99.20003e–29	+9.92000E–28

Examples of invalid floating-point number entries are shown below.

Invalid Entry	Explanation
–433E23	Missing decimal point.
10e-19	Missing decimal point.
10.e19	The mantissa cannot contain spaces between digits or characters. This is accepted as 10.e0, and an error message is displayed.
4.1e19	The exponent cannot contain spaces between digits or characters. This is accepted as 4.1e0, and an error message is displayed.

Errors in Floating-Point Numbers and Operations

On a 352 CPU, overflow occurs when a number greater than 3.402823E+38 or less than -3.402823E+38 is generated by a REAL function. On all other 90-30 models that support floating point operations, the range is greater than 2^{16} or less than $-2^{16.}$ When your number exceeds the range, the ok output of the function is set OFF; and the result is set to positive infinity (for a number greater than 3.402823E+38 on a 352 CPU or 2^{16} on all other models) or negative infinity (for a number less than –3.402823E+38 or -2^{16} on all other models). You can determine where this occurs by testing the sense of the ok output.

POS_INF	= 7F800000h	– IEEE positive infinity representation in hex.
NEG_INF	= FF800000h	– IEEE negative infinity representation in hex.

Note

If you are using software floating point (all models capable of floating point operations except the 352 CPU), numbers are rounded to zero (0) at ±1.175494E–38.

If the infinities produced by overflow are used as operands to other REAL functions, they may cause an undefined result. This undefined result is referred to as an NaN (Not a Number). For example, the result of adding positive infinity to negative infinity is undefined. When the ADD_REAL function is invoked with positive infinity and negative infinity as its operands, it produces an NaN for its result.

On a 352 CPU, each REAL function capable of producing an NaN produces a specialized NaN which identifies the function:

NaN_ADD.	= 7F81FFFFh	– Real addition error value in hex.
NaN_SUB	= 7F81FFFFh	– Real subtraction error value in hex.
NaN_MUL	= 7F82FFFFh	– Real multiplication error value in hex.
NaN_DIV	= 7F83FFFFh	– Real division error value in hex.
NaN_SQRT	= 7F84FFFFh	– Real square root error value in hex.
NaN_LOG	= 7F85FFFFh	– Real logarithm error value in hex.
NaN_POW0	= 7F86FFFFh	– Real exponent error value in hex.
NaN_SIN	= 7F87FFFFh	– Real sine error value in hex.
NaN_COS	= 7F88FFFFh	– Real cosine error value in hex.
NaN_TAN	= 7F89FFFFh	– Real tangent error value in hex.
NaN_ASIN	= 7F8AFFFFh	– Real inverse sine error value in hex.
NaN_ACOS	= 7F8BFFFFh	– Real inverse cosine error value in hex.
NaN_BCD	= 7F8CFFFFh	– BCD-4 to real error.
REAL_INDEF	= FFC00000h	– Real indefinite, divide 0 by 0 error.

All other CPUs that support floating point operations produce one (1) Nan output: FFFF FFFF.

When an NaN result is fed into another function, it passes through to the result. For example, if an NaN_ADD is the first operand to the SUB_REAL function, the result of the SUB_REAL is NaN_ADD. If both operands to a function are NaNs, the first operand will pass through. Because

of this feature of propagating NaNs through functions, you can identify the function where the NaN originated.

Note

For NaN, the ok output is OFF (not energized).

The following table explains when power is or is not passed when dealing with numbers viewed as or equal to infinity. As shown previously, outputs that exceed the positive or negative limits are viewed as POS_INF or NEG_INF respectively.

Operation	Input 1	Input 2	Output	Powerflow
All	Number	Number	Positive or Negative Infinity	No
All Except Division	Infinity	Number	Infinity	Yes
All	Number	Infinity	Infinity	Yes
Division	Infinity	Number	Infinity	No
All	Number	Number	NaN	No

General Case of Power Flow for Floating-Point Operations

3

350 and 360 series CPUs: changing mode with key switch, 2-14
350 and 360 series CPUs: key switch, 2-14

A

ACOS, 4-35
ADD, 4-27
ADD_IOM, 2-25
ADD_SIO, 2-25
Addition function, 4-27
Addition of I/O module, 3-18
Alarm, 3-2
Alarm error codes, B-5
Alarm processor, 3-2
ALT keys, D-1
AND, 4-49
ANY_FLT, 2-26
APL_FLT, 2-25
Application fault, 3-12
Application program logic scan, 2-8
ARRAY_MOVE, 4-87
ASIN, 4-35
ATAN, 4-35

B

BAD_PWD, 2-26
BAD_RAM, 2-26
Base 10 logarithm function, 4-37
Battery signal, low, 3-11
BCD-4, 2-23, 4-95
BCLR, 4-62
BIT, 2-23
Bit clear function, 4-62
Bit operation functions, 4-47
  AND, 4-49
  BCLR, 4-62
  BPOS, 4-64
  BSET, 4-62
  BTST, 4-60
  MCMP, 4-66
  NOT, 4-53
  OR, 4-49
  ROL, 4-58
  ROR, 4-58
  SHL, 4-55
  SHR, 4-55
  XOR, 4-51
Bit position function, 4-64
Bit sequencer function, 4-80
Bit set function, 4-62
Bit test function, 4-60
BITSEQ, 4-80
  memory required, 4-80
BLKCLR, 4-75
BLKMOV, 4-73
Block clear function, 4-75
Block locking feature, 2-37
  EDITLOCK, 2-37
  permanently locking a subroutine, 2-37
  VIEWLOCK, 2-37
Block move function, 4-73
Boolean execution speed, A-10
BPOS, 4-64
BSET, 4-62
BTST, 4-60
BYTE, 2-23

C

CALL, 4-108
Call function, 4-108
CFG_MM, 2-25
Checksum calculation, 2-9
Checksum failure, program block, 3-11
Clocks, 2-34
  elapsed time clock, 2-34
  time-of-day clock, 2-34
Coil
  with Multiple and Single coil checking, 4-6
Coils, 4-3, 4-4
  continuation coil, 4-8
  negated coil, 4-4
  negated retentive coil, 4-5
  negative transition coil, 4-5
  positive transition coil, 4-5
  RESET coil, 4-6
  retentive coil, 4-5
  retentive RESET coil, 4-7
  retentive SET coil, 4-7
  SET coil, 4-6
COMMENT, 4-131
Comment function, 4-131
COMMREQ, 4-83
  error code, description, and correction, 3-11
Communication request function, 4-83
  error code, description, and correction, 3-11
Communication window modes, 2-14
Communications failure during store, 3-16
Communications with the PLC, 2-12
Configuration, 4-1
Configuration mismatch, system, 3-10
Constant sweep time exceeded, 3-12
Constant sweep time mode, 2-13, 2-35
Constant sweep timer, 2-35
Contacts, 4-2

Continuation contact, 4-8
normally closed contact, 4-4
normally open contact, 4-4
Continuation coil, 4-8
Continuation contact, 4-8
Control functions, 4-107
CALL, 4-108
COMMENT, 4-131
DOIO, 4-109
enhanced DOIO for model 331 and higher CPUs, 4-113
END, 4-123
ENDMCR, 4-127
JUMP, 4-128
LABEL, 4-130
MCR, 4-124
PID, 4-171
Sequential Event Recorder, 4-114
SER, 4-114
SVCREQ, 4-132
Control functions Instruction timing, CPU, A-1
Conversion functions, 4-94
BCD-4, 4-95
DINT, 4-99
INT, 4-97
REAL, 4-101
TRUN, 4-105
WORD, 4-103
Convert to BCD-4 function, 4-95
Convert to double precision signed integer function, 4-99
Convert to Real function, 4-101
Convert to signed integer function, 4-97
Convert to Word function, 4-103
Corrupted memory, 3-7
Corrupted user program on power-up, 3-13
COS, 4-35
Cosine function, 4-35
Counters, 4-9
DNCTR, 4-22
function block data, 4-9
UPCTR, 4-20
CPU sweep, 2-2
CTRL keys, D-1

D

Data move functions, 4-69
BITSEQ, 4-80
BLKCLR, 4-75
BLKMOV, 4-73
COMMREQ, 4-83
MOVE, 4-70
SHFR, 4-77
Data retentiveness, 2-21
Data types, 2-23
BCD-4, 2-23
BIT, 2-23
BYTE, 2-23
DINT, 2-23
INT, 2-23
REAL, 2-23
WORD, 2-23
Defaults conditions for model 30 output modules, 2-41
DEG, 4-39
Diagnostic data, 2-41
Diagnostic faults, 3-4
addition of I/O module, 3-18
application fault, 3-12
constant sweep time exceeded, 3-12
loss of I/O module, 3-17
loss of, or missing, option module, 3-8
low battery signal, 3-11
reset of, addition of, or extra, option module, 3-9
DINT, 2-23, 4-99
Discrete references, 2-20
discrete inputs, 2-20
discrete internal, 2-20
discrete outputs, 2-20
discrete temporary, 2-21
global data, 2-21
system references, 3-5
system status, 2-21, 2-24
DIV, 4-27
Division function, 4-27
DNCTR, 4-22
Do I/O function, 4-109
enhanced DO I/O function for model 331 and higher CPUs, 4-113
DOIO, 4-109
enhanced DOIO for model 331 and higher CPUs, 4-113
Double precision signed integer, 2-23
Down counter, 4-22

E

EDITLOCK, 2-37
Elapsed time clock, 2-34
END, 4-123
End function, 4-123
End master control relay function, 4-127
ENDMCR, 4-127
Enhanced DO I/O function for the model 331 and higher CPUs, 4-113
EQ, 4-41
Equal function, 4-41
Error codes, B-5

EXP, 4-37
Exponential functions, 4-37
  power of e, 4-37
  power of X, 4-37
EXPT, 4-37
External I/O failures, 3-2

F

Fatal faults, 3-4
  communications failure during store, 3-16
  corrupted user program on power-up, 3-13
  option module software failure, 3-11
  PLC CPU system software failure, 3-14
  program block checksum failure, 3-11
  system configuration mismatch, 3-10
Fault action, 3-4
  diagnostic faults, 3-4
  fatal faults, 3-4
  I/O fault action, B-11
  informational faults, 3-4
  PLC fault action, B-5
fault actions, 3-8
Fault category, 3-17
Fault description, 3-17
Fault effects, additional, 3-5
Fault expanations and correction, 3-1
Fault explanation and correction
  I/O fault group, B-10
  interpreting a fault, B-1
  PLC fault group, B-4
Fault explanations and correction
  accessing additional fault information, 3-6
  addition of I/O module, 3-18
  application fault, 3-12
  communications failure during store, 3-16
  constant sweep time exceeded, 3-12
  corrupted user program on power-up, 3-13
  fault category, 3-17
  fault description, 3-17
  fault handling, 3-2
  fault type, 3-17
  I/O fault table, 3-5
  I/O fault table explanations, 3-17
  loss of I/O module, 3-17
  loss of, or missing, option module, 3-8
  low battery signal, 3-11
  no user program present, 3-13
  non-configurable faults, 3-8
  option module software failure, 3-11
  password access failure, 3-13
  PLC CPU system software failure, 3-14
  PLC fault table, 3-5
  PLC fault table explanations, 3-7
  program block checksum failure, 3-11
  reset of, addition of, or extra, option module, 3-9
  system configuration mismatch, 3-10
Fault group, B-4, B-10
Fault handling, 3-2
  alarm processor, 3-2
  fault action, 3-4
Fault references, 3-4
  definitions of, 3-5
Fault type, 3-17
Faults, 3-2
  accessing additional fault information, 3-6
  actions, 3-8
  addition of I/O module, 3-18
  additional fault effects, 3-5
  application fault, 3-12
  classes of faults, 3-2
  communications failure during store, 3-16
  constant sweep time exceeded, 3-12
  corrupted user program on power-up, 3-13
  error codes, B-5
  explanations and correction, 3-1
  external I/O failures, 3-2
  fault action, 3-4
  I/O fault action, B-11
  I/O fault group, B-10
  I/O fault table, 3-3, 3-5
  I/O fault table explanations, 3-17
  internal failures, 3-2
  interpreting a fault, B-1
  loss of I/O module, 3-17
  loss of, or missing, option module, 3-8
  low battery signal, 3-11
  no user program present, 3-13
  operational failures, 3-2
  option module software failure, 3-11
  password access failure, 3-13
  PLC CPU system software failure, 3-14
  PLC fault action, B-5
  PLC fault group, B-4
  PLC fault table, 3-3, 3-5
  PLC fault table explanations, 3-7
  program block checksum failure, 3-11
  references, 3-4
  reset of, addition of, or extra, option module, 3-9
  system configuration mismatch, 3-10
  system reaction to faults, 3-3
Faults, interpreting, B-1
Flash protection on 350 and 360 series CPUs, 2-14
Floating-point numbers, E-1
  entering and displaying floating-point numbers, E-5
  errors in floating-point numbers and operations, E-6
  internal format of floating-point numbers, E-3
  values of floating-point numbers, E-4
Function block parameters, 2-28

Function block structure, 2-26
format of program function blocks, 2-27
format of relays, 2-26
function block parameters, 2-28
power flow, 2-29

G

GE, 4-41
Global data, 2-42
Global data references, 2-21
Greater than function, 4-41
Greater than or equal function, 4-41
GT, 4-41

H

Horizontal link, 4-7
Housekeeping, 2-6
HRD_CPU, 2-25
HRD_FLT, 2-26
HRD_SIO, 2-25

I

I/O data formats, 2-41
I/O fault table, 3-3, 3-5, B-8
explanations, 3-17
fault action, B-11
fault actions for specific faults, B-11
fault address, B-9
fault group, B-10
fault specific data, B-11
fault time stamp, B-12
interpreting a fault, B-1
long/short indicator, B-9
point, B-10
rack, B-10
reference address, B-9
slot, B-10
symbolic fault specific data, B-11
I/O structure, Series 90-30 PLC, 2-38
I/O system , Series 90-30 PLC, 2-38
I/O system, Series 90- Micro PLC
Micro I/O, 2-43
I/O system, Series 90-20 PLC, 2-38
model 20 I/O modules, 2-42
I/O system, Series 90-30 PLC
default conditions for model 30 output modules, 2-41
diagnostic data, 2-41
global data, 2-42
I/O data formats, 2-41
model 30 I/O modules, 2-39
Informational faults, 3-4
no user program present, 3-13
password access failure, 3-13
Input references, discrete, 2-20
Input register references, analog, 2-20
Input scan, 2-7
Instruction mnemonics, C-1
Instruction set, 4-1
bit operation functions, 4-47
control functions, 4-107
conversion functions, 4-94
data move functions, 4-69
math functions, 4-26
relational functions, 4-41
relay functions, 4-2
table functions, 4-86
timers and counters, 4-9
Instructions, programming, 4-1
bit operation functions, 4-47
control functions, 4-107
conversion functions, 4-94
data move functions, 4-69
instruction mnemonics, C-1
math functions, 4-26
relational functions, 4-41
relay functions, 4-2
table functions, 4-86
timers and counters, 4-9
INT, 2-23, 4-97
Internal failures, 3-2
Internal references, discrete, 2-20
Inverse cosine function, 4-35
Inverse sine function, 4-35
Inverse tangent function, 4-35
IO_FLT, 2-26
IO_PRES, 2-26

J

JUMP, 4-128
Jump instruction, 4-128

K

Key switch on 350 and 360 Series CPUs, 2-14

L

LABEL, 4-130
Label instruction, 4-130
LE, 4-41
Less than function, 4-41
Less than or equal function, 4-41
Levels, privilege, 2-36
change requests, 2-37

Links, horizontal and vertical, 4-7
LN, 4-37
Locking/unlocking subroutines, 2-37
LOG, 4-37
Logarithmic functions, 4-37
  base 10 logarithm, 4-37
  natural logarithm, 4-37
Logic program checksum calculation, 2-9
Logic solution, 2-8
Logical AND function, 4-49
Logical NOT function, 4-53
Logical OR function, 4-49
Logical XOR function, 4-51
LOS_IOM, 2-25
LOS_SIO, 2-25
Loss of I/O module, 3-17
Loss of, or missing, option module, 3-8
Low battery signal, 3-11
LOW_BAT, 2-25
LT, 4-41

M

Maintenance, 3-1
Masked compare function, 4-66
Master control relay function, 4-124
Math functions, 4-26
  ACOS, 4-35
  ADD, 4-27
  ASIN, 4-35
  ATAN, 4-35
  COS, 4-35
  DEG, 4-39
  DIV, 4-27
  EXP, 4-37
  EXPT, 4-37
  LN, 4-37
  LOG, 4-37
  MOD, 4-31
  MUL, 4-27
  RAD, 4-39
  SIN, 4-35
  SQRT, 4-33
  SUB, 4-27
  TAN, 4-35
MCR, 4-124
Memory, corrupted, 3-7
Micro Models, 2-43
Mnemonics, instruction, C-1
MOD, 4-31
Model 20 I/O modules, 2-42
Model 30 I/O modules, 2-39
Modulo function, 4-31
MOVE, 4-70
Move function, 4-70
MSKCMP, 4-66
MUL, 4-27
Multiplication function, 4-27

N

Natural logarithm function, 4-37
NE, 4-41
Negated coil, 4-4
Negated retentive coil, 4-5
Negative transition coil, 4-5
No user program present, 3-13
Normally closed contact, 4-4
Normally open contact, 4-4
NOT, 4-53
Not equal function, 4-41

O

OFDT, 4-17
Off-delay timer, 4-17
On-delay timer, 4-11, 4-14
ONDTR, 4-11
Operation of system, 2-1
Operational failures, 3-2
Option module software failure, 3-11
OR, 4-49
Output references, discrete, 2-20
Output register references, analog, 2-20
Output scan, 2-9
OV_SWP, 2-25
Overrides, 2-21

P

Password access failure, 3-13
Passwords, 2-36
PB_SUM, 2-25
PCM communications with the PLC, 2-12
Periodic subroutines, 2-19
PID, 4-171
PLC CPU system software failure, 3-14
PLC fault table, 3-3, 3-5, B-2
  error codes, B-5
  explanations, 3-7
  fault action, B-5
  fault extra data, B-7
  fault group, B-4
  fault time stamp, B-7
  interpreting a fault, B-1
  long/short indicator, B-3
  rack, B-3

slot, B-3
spare, B-3
task, B-3
PLC sweep, 2-2
application program logic scan, 2-8
configured constant sweep time mode, 2-13
constant sweep time mode, 2-13, 2-35
housekeeping, 2-6
input scan, 2-7
logic program checksum calculation, 2-9
logic solution, 2-8
output scan, 2-9
PCM communications with the PLC, 2-12
programmer communications window, 2-9
scan time contributions for 350 and 360 Series of, 2-5
standard program sweep mode, 2-2
standard program sweep variations, 2-13
STOP mode, 2-13
sweep time calculation, 2-6
sweep time contribution, 2-4
system communications window, 2-10
PLC system operation, 2-1
Positive transition coil, 4-5
Power flow, 2-29
Power of e function, 4-37
Power of X function, 4-37
Power-down, 2-33
Power-up, 2-30
Power-up and power-down sequences, 2-30
power-down, 2-33
power-up, 2-30
Privilege level change requests, 2-37
Privilege levels, 2-36
change requests, 2-37
Program block
how blocks are called, 2-19
how C blocks are called, 2-19
how subroutines are called, 2-19
subroutine block, 2-16
Program block checksum failure, 3-11
Program organization and user data
floating-point numbers, E-1
Program organization and user references/data, 2-16
data types, 2-23
function block structure, 2-26
retentiveness of data, 2-21
system status, 2-24
transitions and overrides, 2-21
user references, 2-20
Program structure
how blocks are called, 2-19
how C blocks are called, 2-19
how subroutines are called, 2-19
subroutine block, 2-16
Program sweep, standard, 2-2
Programmer communications window, 2-9
Programming instructions, 4-1
bit operation functions, 4-47
control functions, 4-107
conversion functions, 4-94
data move functions, 4-69
instruction mnemonics, C-1
math functions, 4-26
relational functions, 4-41
relay functions, 4-2
table functions, 4-86
timers and counters, 4-9
Proportional Integral Deviation (PID), 4-171

R

RAD, 4-39
Radian conversion function, 4-39
RANGE, 4-44
Range function, 4-44
REAL
convert to REAL, 4-101
Data type structure, 2-23
Using floating-point numbers, E-1
Using Real numbers, E-1
references, 2-20
Register Reference
system registers, 2-20
Register references, 2-20
analog inputs, 2-20
analog outputs, 2-20
Relational functions, 4-41
EQ, 4-41
GE, 4-41
GT, 4-41
LE, 4-41
LT, 4-41
NE, 4-41
RANGE, 4-44
Relay functions, 4-2
coils, 4-3, 4-4
contacts, 4-2
continuation coil, 4-8
continuation contact, 4-8
horizontal and vertical links, 4-7
negated coil, 4-4
negated retentive coil, 4-5
negative transition coil, 4-5
normally closed contact, 4-4
normally open contact, 4-4
positive transition coil, 4-5
RESET coil, 4-6
retentive coil, 4-5
retentive RESET coil, 4-7
retentive SET coil, 4-7

SET coil, 4-6
RESET coil, 4-6
Reset of, addition of, or extra, option module, 3-9
Retentive coil, 4-5
Retentive RESET coil, 4-7
Retentive SET coil, 4-7
Retentiveness of data, 2-21
ROL, 4-58
ROR, 4-58
Rotate left function, 4-58
Rotate right function, 4-58

S

Scan Time Contributions for 350 and 360 Series of CPUs, 2-5
Scan, input, 2-7
Scan, output, 2-9
Search array move function, 4-87
Search equal function, 4-91
Search greater than function, 4-91
Search greater than or equal function, 4-91
Search less than function, 4-91
Search less than or equal function, 4-91
Search not equal function, 4-91
Security, system, 2-36
  locking/unlocking subroutines, 2-37
  passwords, 2-36
  privilege level change requests, 2-37
  privilege levels, 2-36
Sequential Event Recorder, 4-114
SER, 4-114
SER function, 4-114
Series 90- Micro PLC I/O system
  Micro CPU and I/O, 2-43
Series 90-20 PLC I/O system, 2-38
  model 20 I/O modules, 2-42
Series 90-30 PLC I/O system, 2-38
  default conditions for model 30 output modules, 2-41
  diagnostic data, 2-41
  global data, 2-42
  I/O data formats, 2-41
  I/O structure, 2-38
  model 30 I/O modules, 2-39
Service Request
  change/read number of words to checksum, 4-143
  change/read time-of-day clock, 4-145
  clear fault tables, 4-155
  Fast Backplane Status Access, 4-165
  interrogate I/O, 4-163
  read elapsed power down time, 4-164
  read elapsed time clock, 4-160
  read I/O override status, 4-161
  read last-logged fault table entry, 4-156
  read master checksum, 4-162
  shut down (stop) PLC, 4-154
Service request function, 4-132
SET coil, 4-6
SFT_CPU, 2-26
SFT_FLT, 2-26
SFT_SIO, 2-25
SHFR, 4-77
Shift left function, 4-55
Shift register function, 4-77
Shift right function, 4-55
SHL, 4-55
SHR, 4-55
Signed integer, 2-23
SIN, 4-35
Sine function, 4-35
SNPX_RD, 2-25
SNPX_WT, 2-25
SNPXACT, 2-25
Software failure, option module, 3-11
SQRT, 4-33
Square root function, 4-33
SRCH_EQ, 4-91
SRCH_GE, 4-91
SRCH_GT, 4-91
SRCH_LE, 4-91
SRCH_LT, 4-91
SRCH_NE, 4-91
Standard program sweep mode, 2-2
Standard program sweep variations, 2-13
Status references, system, 2-21, 2-24
STOP mode, 2-13
STOR_ER, 2-26
SUB, 4-27
Subroutine blocks, 2-16
Subroutines, locking/unlocking, 2-37
Subtraction function, 4-27
SVCREQ, 4-132
  change programmer communications window (#3), 4-139
  change system communications window (#4), 4-141
  change/read constant sweep timer (#1), 4-134
  change/read number of words to checksum, 4-143
  change/read time–of–day clock, 4-145
  clear fault table, 4-155
  Fast Backplane Status Access, 4-165
  interrogate I/O, 4-163
  read elapsed power down time, 4-164
  read elapsed time clock, 4-160

read folder name (#10), 4-151
read I/O override status, 4-161
read last–logged fault table entry, 4-156
read master checksum, 4-162
read PLC ID (#11), 4-152
read PLC run state (#12), 4-153
read sweep time (#9), 4-150
read window values (#2), 4-137
reset watchdog timer (#8), 4-149
shut down (stop) PLC, 4-154
Sweep time calculation, 2-6
Sweep, PLC, 2-2
application program logic scan, 2-8
constant sweep time mode, 2-13, 2-35
housekeeping, 2-6
input scan, 2-7
logic program checksum calculation, 2-9
logic solution, 2-8
output scan, 2-9
PCM communications with the PLC, 2-12
programmer communications window, 2-9
scan time contributions for 350 and 360 Series of CPUs, 2-5
standard program sweep mode, 2-2
standard program sweep variations, 2-13
STOP mode, 2-13
sweep time calculation, 2-6
sweep time contribution, 2-4
system communications window, 2-10
SY_FLT, 2-26
SY_PRES, 2-26
System communications window, 2-10
System configuration mismatch, 3-10
System operation, 2-1
clocks and timers, 2-34
PLC sweep summary, 2-2
power-up and power-down sequences, 2-30
program organization and user references/data, 2-16
Series 90-20 PLC I/O system, 2-38
Series 90-30 PLC I/O system, 2-38
system security, 2-36
System references, 3-5
System register references, 2-20
System status references, 2-21, 2-24
ADD_IOM, 2-25
ADD_SIO, 2-25
ANY_FLT, 2-26
APL_FLT, 2-25
BAD_PWD, 2-26
BAD_RAM, 2-26
CFG_MM, 2-25
HRD_CPU, 2-25
HRD_FLT, 2-26
HRD_SIO, 2-25
IO_FLT, 2-26
IO_PRES, 2-26
LOS_IOM, 2-25
LOS_SIO, 2-25
LOW_BAT, 2-25
OV_SWP, 2-25
PB_SUM, 2-25
SFT_CPU, 2-26
SFT_FLT, 2-26
SFT_SIO, 2-25
SNPX_RD, 2-25
SNPX_WT, 2-25
SNPXACT, 2-25
STOR_ER, 2-26
SY_FLT, 2-26
SY_PRES, 2-26

T

Table functions, 4-86
ARRAY_MOVE, 4-87
search less than or equal function, 4-91
SRCH_EQ, 4-91
SRCH_GE, 4-91
SRCH_GT, 4-91
SRCH_LT, 4-91
SRCH_NE, 4-91
TAN, 4-35
Tangent function, 4-35
Temporary references, discrete, 2-21
Time-of-day clock, 2-34
Timers, 2-34, 4-9
constant sweep timer, 2-35
function block data, 4-9
OFDT, 4-17
ONDTR, 4-11
time–tick contacts, 2-35
TMR, 4-14
Watchdog timer, 2-35
Time-tick contacts, 2-35
Timing, instruction, A-1
TMR, 4-14
Transitions, 2-21
Troubleshooting, 3-1
accessing additional fault information, 3-6
I/O fault table, 3-5
I/O fault table explanations, 3-17
interpreting a fault, B-1
non-configurable faults, 3-8
PLC fault table, 3-5
PLC fault table explanations, 3-7
TRUN, 4-105
Truncate function, 4-105

U

Up counter, 4-20
UPCTR, 4-20

User references, 2-20
analog inputs, 2-20
analog outputs, 2-20
discrete inputs, 2-20
discrete internal, 2-20
discrete outputs, 2-20
discrete references, 2-20
discrete temporary, 2-21
global data, 2-21
register references, 2-20
system references, 3-5
system registers, 2-20
system status, 2-21, 2-24

V

Vertical link, 4-7
VIEWLOCK, 2-37

W

Watchdog timer, 2-35
Window, 2-9
programmer communications window, 2-9
system communications window, 2-10
WORD, 2-23, 4-103

X

XOR, 4-51